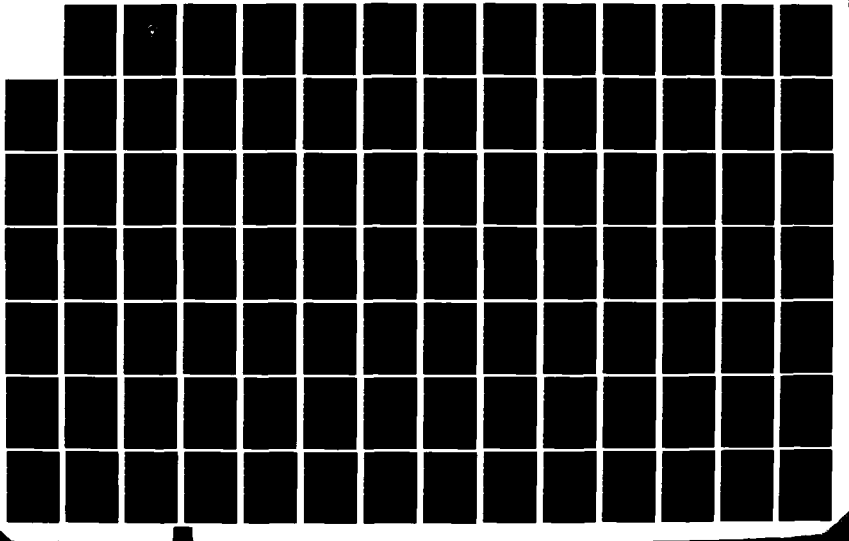
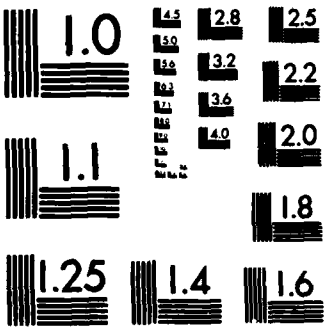
AD-A127 417 A HEURISTIC FOR DECOMPOSING A PROBLEM INTO A SEQUENCE OF SUBPROBLEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D V EVANS DEC 82 1/2

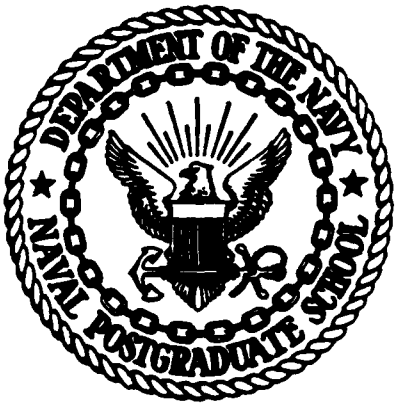
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

A HEURISTIC FOR DECOMPOSING A PROBLEM INTO A SEQUENCE OF SUBPROBLEMS

by

Donald Vincent Evans
December 1982

Thesis Advisor: D. R. Smith

Approved for Public Release; Distribution Unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A127417
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) A Heuristic for Decomposing a Problem into a Sequence of Subproblems	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December, 1982
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s) Donald Vincent Evans	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	12. REPORT DATE December, 1982
	13. NUMBER OF PAGES 103
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Program Synthesis; Automatic Program Synthesis; Top-down Program Synthesis; Decomposing Problems into Subproblems; Artificial Intelligence

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis presents a method for decomposing a specification of a problem into a sequence of subproblem specifications. The method uses the specification to build a tree-like structure called a semantic net. The net is then used to construct a sequence of subspecifications. Each subspecification of the sequence represents a subproblem. Composition of the solution to the subproblems results in a solution to the given problem specification.

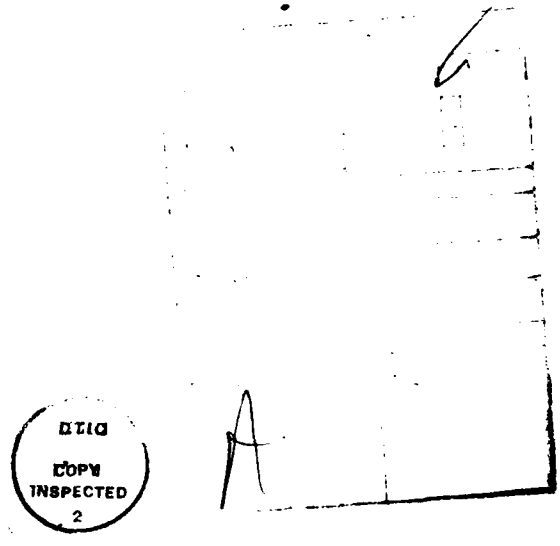
(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued) Block # 20

In this work, the author we present an intuitive approach to what Artificial Intelligence and program synthesis is, define the sequence problem associated with program synthesis, and present the method for deriving a sequence of subspecifications. When this has been done, the method is then applied to a specific problem domain called the Blocks World. We then consider the method in a non-Blocks World domain and follow with a summary.



Approved for public release; distribution unlimited

A Heuristic for Decomposing a Problem into a Sequence of Subproblems

by

Donald Vincent Evans
Captain, United States Marine Corps
B.S., Southern University and A&M College, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
December 1982

Author: Donald V. Evans

Approved by: Douglas R. Smith
Thesis Advisor

Bruce J. MacLennan
Second Reader

Harold K. Horton
Chairman, Department of Computer Science

W M Woods
Dean of Information and Policy Sciences

ABSTRACT

This thesis presents a method for decomposing a specification of a problem into a sequence of subproblem specifications. The method uses the specification to build a tree-like structure called a semantic net. The net is then used to construct a sequence of subspecifications. Each subspecification of the sequence represents a subproblem. Composition of the solutions to the subproblems results in a solution to the given problem specification.

In this work, we present an intuitive approach to what Artificial Intelligence and program synthesis is, define the sequence problem associated with program synthesis, and present the method for deriving a sequence of subspecifications. When this has been done, the method is then applied to a specific problem domain called the Blocks World. We then consider the method in a non-Blocks World domain and follow with a summary.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

To my advisors, Professors Smith and MacLennan, thank you for helping me with this work.

To Chris Nance, Beulah Canion and members of First Baptist Church, Pacific Grove, California, thank you for your support and prayers.

To Professor Harold Fredricksen, thank you for your superb help during my studies at NPS.

To my loving wife, Estella, who endured all the hardships that a time like this brings to a family, thank you.

I. INTRODUCTION

What is programming? Is programming really a creative activity reserved only for human participation? What is the essence, the bare bones of it, that distinguishes it from other activities or things? As we ponder these questions, let us observe the relationship between the expert and the apprentice. The expert is busily working away on the job and then he notices his helper standing idly on the side. The expert then says to his helper, "Do this and do this and do this and then do that." And now, we can observe the helper being as busy as the expert. Both are being constructive. Clearly, the "how to do it" was left up to the helper. But the question we might ask is, "Was the helper programmed by the expert?"

In the past fifteen to 25 years, a small number of theoretical psychologists, mathematicians, and computer scientists in the area of Artificial Intelligence have started to equip computers with the ability to perform programming. We call this ability automatic programming when referring to the computer being able to write programs. It is dubbed automatic because the programming is accomplished without human intervention once the specification is supplied to it. The specification is analogous to the information given by the expert to the helper in the above scenario.

The above questions and scenario involving the expert and apprentice are an attempt to show that programming is a complex subject. This thesis will focus upon a small but significant ingredient of the programming process, the sequence. The aim of this thesis is similar to what transpired between the expert and the apprentice above. We want to develop a set of methods that will consider "in toto" the set of events that describe a problem and then determine a sequence for "achieving" the events, if one exists, that will realize a solution to the problem.

This thesis considers the "sequence problem" as it relates to automatic program synthesis. In so doing, we present a novel method for decomposing a problem into a sequence of subproblems. The method uses the specification to build a tree-like structure called a semantic net to determine a sequence of specifications. Since automatic program synthesis falls within the area of Artificial Intelligence and since Artificial Intelligence means different things to different people, we find it best to proceed by first establishing intuitions and frames of reference about the concepts used to present a solution method for this problem.

II. NATURE OF ARTIFICIAL INTELLIGENCE

When the term "Artificial Intelligence" (the capitalized form - henceforth AI) is used, the emphasis is on the subarea of Computer Science; a subfield of a technology. The uncapitalized form, "artificial intelligence", on the other hand, refers to the mechanisms or things that AI attempts to discover. As such, we can say AI attempts to discover artificial intelligence.

The nature of AI can be understood best by observing its primary activity, its problem domain, its goal, and its motive for doing research. The primary activity of AI is empirical investigation. AI has few, if any, theories. Consequently, investigative strategies have been substituted for the missing theories. Additionally, AI's problem domain is complex and only partially understood at best. It follows therefore that highly structured or tailored techniques do not exist. In short, one can view AI as the area where problems are not fully understood, are yet solvable by humans in most cases, and have no special-purpose techniques for providing a solution. Simon in [1] defines AI as:

> "The domain in which it has not yet been possible to substitute powerful special-purpose techniques for weak methods. At any time that such techniques are discovered for a particular subset of problems, those problems are removed from the jurisdiction of AI to that of operations research or numerical analysis."

We shall in this thesis also adopt this view of AI. Aside from noticing the dynamics of Simon's definition, it becomes evident, if his definition is true, that AI will always seem to be groping along on the frontier (and pushing the state-of-the-art!).

The goal of AI is to equip machines with a human-like ability to solve problems [2]. This goal is best understood within the context of the evolution of our demand upon computers. Initially, the scope of the demand was to have computers handle large, simple and highly redundant jobs like census tabulations and projectile trajectory calculations. Currently, supercomputers are used on extremely large and complex flow-rate problems like those associated with meteorology. Researchers are also exploring computer usage in the decision-making processes. These systems are known as Decision Support Systems (DSS). An interesting performance criterion of a DSS is its ability to support not only different decision-making processes but also support a variety of cognitive styles [3]. This performance criterion is an example of the newer kinds of demand we are now placing on computers. Two other examples are the increasing desire to interface with a database using human speech, and the increasing desire to employ autonomous robots in high risk situations like mining or military applications.

The importance of these examples is that some computer experts feel that the kind of computing associated with AI will dominate the kind of computing we do in the future. For example, during a private conversation in October 1982, R. W. Hamming summarized the evolution of computing by Figure 2.1. The figure shows the doubling of computer capacity over time as a result of the kinds of computing. Each curve in Figure 2.1 represents the kind of computing that summarizes user demands over some time frame. Note how AI is predicted as the next classification of user demand. Consequently, if this be true, what we do or fail to do in AI greatly effects our ability to cope with our problems of the near and distant future.

As a result, the motive for doing AI research is precisely our aggregate demand on computers that gets translated into users wanting to have computers perform like humans in less well-structured problem domains. Ironically, the motives for doing the research make a suitable definition for AI - the area of discovering mechanisms, for computer usage, that enable a system like the human mind to behave purposefully, adaptively, and sometimes even effectively, over a wide range of difficult and ill-structured tasks [2]. Probably the most striking effort (i.e., comprehensive, ambitious, coordinated, etc.) is the Japanese proposal for the fifth generation computer [4].

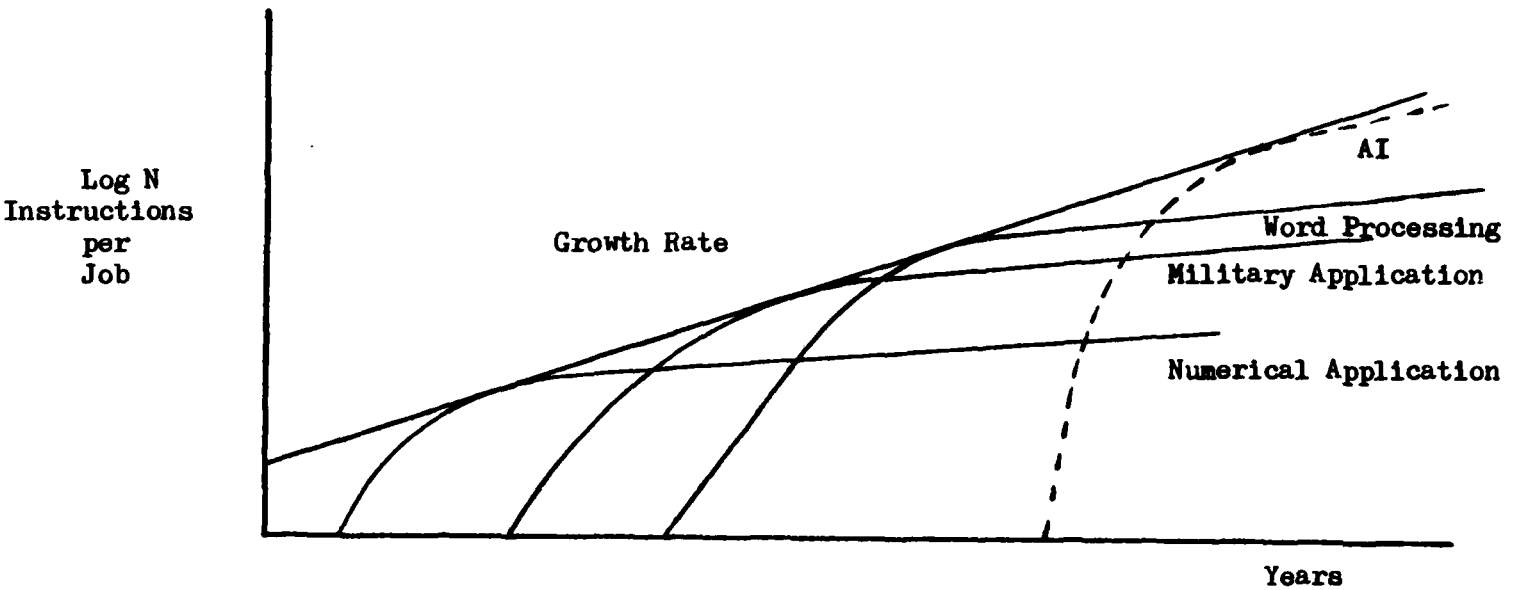


Figure 2.1

Hamming's Summary of User Demand

III. NATURE OF PROGRAM SYNTHESIS

A. GENERAL

The terms "program synthesis" and "automatic program synthesis" are used interchangeably, but one should keep in mind that the aim is to equip the computer with the skill to do programming.

Formally, program synthesis can be defined as the art of deriving a program from a given problem specification without specifying any algorithm [5]. Simply, the requirements that the program must meet are stated without indicating how to create the program. The "how" is the jurisdiction of the synthesis system. Regardless of the means for doing the synthesis, by human or machine, a program synthesis system must be provided with these basic things:

- a specification which is capable of stating a problem which the synthesis system has been designed to solve;
- knowledge for coordinating the synthesis system's actions as it proceeds to transform the problem into a program that is a solution to the specification;
- an ability to produce code for the target language.

Again, this thesis is concerned with one aspect of the second basic requirement, namely the "sequence problem" of program synthesis.

The relationship of this work to program synthesis is the following. Smith in [6] describes the structure of a top-down program synthesis system. The system is logically composed of two parts, namely the Programming Knowledge Base and the Synthesis Control. The Programming Knowledge Base consists of two parts, the Data Structure Knowledge Base and the Library of Design Methods. The Library of Design Methods contains design theories for various classes of algorithms. These design theories break a nonprimitive problem into subproblems and assemble a solution based on solutions to subproblems. The method we present is concerned with decomposing a problem into a sequence of subproblems and, therefore, would be contained in the Library of Design Methods.

B. BASIC CONCEPTS

The basic requirements of a synthesis system, as stated above, are best understood if the program synthesis process is oversimplified. However, several concepts are needed to do this. First, let us consider the term "specification". The specification is a set of relationships between input values and output values. This kind of specification is known as a procedural abstraction [7]. It has two parts: an interface specification and a behavioral specification. The interface

specification is concerned with things like the module name, parameters and resources for input and output. It is omitted from any further discussion and assumed to be implicit for the synthesis system. The behavioral specification, however, is the portion of the specification that is helpful in understanding program synthesis. The technique for specifying the behavioral specification is the input/output approach. "The input/output approach describes the relationship between the inputs and the outputs by giving a pair of constraints. Provided that the actual input satisfies the input constraints, the output is guaranteed to satisfy the output constraints" [7].

The input portion of the specification is associated with the start state - a description of the current world view defined by relationships between objects or things that have a bearing on the problem. The output portion of the specification is associated with the goal description - a description of relationships between objects that, if established, represent a solution to the problem. These relationships usually correspond to boolean tests within the reasoning language of the synthesis system. These tests comprise an important portion of the knowledge that was alluded to as being a basic requirement of a synthesis system.

The next important term is "operator". An operator is a task-specific mechanism that accomplishes a specific set of actions. The actions correspond to establishing relationships

that transform the current world view into another view. The relationships are established between the objects that somehow come into play with establishing the desired goal state. Since operators are designed to achieve specific results, an operator can only be applied to particular kinds of objects. That is, a particular operator is applicable over a certain set of objects. Additionally, before the operator can be applied to an object, the object must meet certain criteria. The criteria are usually viewed as a list of properties. The properties can be, among other things, relationships between objects. Further, an operator corresponds to a block of code in the target language of the synthesis system that establishes the relationships between objects in the synthesized program.

We are now in a position to discuss more fully the idea of protected relationships. Protected relationships are associated with subgoals in the following manner. Subgoals are accomplished by applying the appropriate operators to the appropriate arguments in an appropriate sequence. As the operators are being applied to the particular arguments, other conditions (relationships) are being established. From those conditions, some set of conditions is needed to show that the subgoal has been established (made to be true). We call that set the protected relationships of the subgoal.

Waldinger in [8] has shown that protected relationships, once established, should not be temporarily undone (made

false) because an infinite sequence of actions for the synthesis system could result. This infinite sequence of actions is possible when the synthesis system considers the achievement of subgoals to be independent from each other when, in fact, they are dependent upon each other. As one subgoal is achieved, it must be undone in order to achieve some other subgoal. But when the other subgoal is achieved it must be undone to achieve the first, etc. When such a relationship between subgoals exist we say that the problem is a nonlinear problem. After achieving the subgoal, then the set of relationships that correspond to the subgoal's protected relationships are determined and saved by some mechanism of the program synthesis system. The subgoal is then called a protected subgoal. In short, the synthesis system, in its reasoning language, "knows" the effect of applying an operator to its arguments.

So then, in the process of establishing other subgoals, the protected relationships of previously established subgoals may be made false or contradicted. This we refer to as a violation or contradiction. When a contradiction is made, a reordering of the sequence in which the subgoals are established is done. If there is a sequence for which no contradiction occurs, then we say that that sequence of operators produces valid results. Otherwise, we conclude that there is no solution to the problem to which the subgoals belong.

Lastly, there is the matter of notation. We adopt the convention of using the terms "WHERE" and "FIND" to indicate the input and output portions of the specification, respectively. Additionally, each syntactic portion of the specification that represents a predicate is called a subgoal and is represented as: (subgoal). Therefore, the problem specification can be represented as a series of smaller goals. The general format adopted in this thesis is:

[(WHERE (s1)(s2) - - - (sm))(FIND (g1)(g2) - - - (gn))] .

This format follows from the above description of the input/output approach. The WHERE portion describes the relevant relationships in the current world and the FIND portion describes the relationships that, if established, represent a solution. The (si)s state relationships in the current world view. The (gi)s state the relationships that constitute a solution. The semantics of the specification is given as follows: IF s1 and s2 and - - and sm hold initially THEN g1 and g2 and - - and gn hold after program execution. Further, for the convenience of distinguishing between tasks that need to be accomplished and those already accomplished, a subgoal in uppercase symbols, (SUBGOAL), represents an accomplished task. The terms "true", "established", "achieved" and "accomplished' are used interchangeably and are used to mean that some (subgoal) has been transformed or synthesized into (SUBGOAL).

C. A SIMPLIFIED EXAMPLE

Now let us consider a simplified program synthesis example. Suppose our goal after waking in the morning is to do personal grooming, get to work, and start our job. Further, suppose the problem is specified as:

```
[(WHERE (PERSON IS-AWAKE)(PERSON IS-UNGROOMED)
     (FIND (person is-dry at work)
           (person is-groomed before working)
           (person is-dressed before arriving at job)
           (person is-working on job))].
```

As the synthesis system begins to formulate a solution to this problem, the *WHERE* portion of the specification is assumed to be established. This represents the starting point or start state for the synthesis system. The system then uses its knowledge to transform the start state until all the subgoals in the FIND portion of the specification are true. When all the conditions of the FIND portion are true, the system has "synthesized" a solution.

Let us go into more detail using the above example. Suppose the synthesis system has the following operators: (brush teeth), (drive to job), (dress), (shower), (shave), and (start job). The overall job of the synthesis system is to apply these operators in such a manner that the FIND portion is true. Here we get a glimpse at the nature of the "sequence

problem" in program synthesis. We see that some sequences satisfy our problem and some do not. For example, one possible solution to the above specification might be (SHOWER)(SHAVE)(BRUSH TEETH)(DRESS)(DRIVE TO JOB)(START JOB). We also note that there are other possible solutions. For example, two of them are:

(SHAVE)(SHOWER)(BRUSH TEETH)(DRESS)(DRIVE TO JOB)(START JOB)

and

(BRUSH TEETH)(SHAVE)(SHOWER)(DRESS)(DRIVE TO JOB)(START JOB).

In fact, the general form of a solution could be: ----(SHOWER)---(DRESS)---(DRIVE TO JOB)---(START JOB). The dashed lines indicate that any of the other operators could be applied at that point; as long as each of the listed operators occurred in the given sequence. This constrains a solution so that a person is dry and dressed before getting to work.

Several important concepts can been seen from this example. First, there may be many solutions which satisfy the specification. This leads to the concept of acceptable versus unacceptable solutions. To ensure that a solution is an acceptable one, the specification must constrain the problem in such a manner that the synthesis system provides the desired one. In the above example, getting to work and then brushing ones teeth is an acceptable activity. If we review the input/output specification, being fully groomed before starting to work is not a constraint. Therefore, brushing

after arriving at work is an acceptable solution. We can see that the specification must be stated in such a manner that all acceptable solutions are allowed, but no others.

Another important concept is the semantics of applying an operator to its arguments. Consider the task (shave) for instance. We must remember that tasks are associated with actions. That is, a task requires that some test be done to establish the existence of some relationship and/or requires that some relationship be established. Therefore, a task or subgoal is accomplished by applying operators to arguments. Continuing with (shave), for example, if PICK-UP was an operator then it might be applied to the arguments HAND and RAZOR, causing the HAND to grasp the RAZOR. But whatever operator is used to perform "shaving", we would like for it to have a different meaning (result) for the argument of females than it does for the argument of males. The set of circumstances surrounding the application of an operator is referred to as its context. We see then that the context affects the application of the operator. Remember, physically, the operator is a block of machine code. Therefore, in general, if an operator has many contexts then these contexts are embedded within the code that defines the operator. For example, we specify "what" (female or male) is to be shaved and the operator acts accordingly.

Additionally, the effect that an operator has upon its argument(s) is equally important. The synthesis system must

have the ability to know this. Using our example, again, the synthesis system must be able to determine that the sequence (DRESS) followed by (SHOWER) causes clothing to be wet and, therefore, an unacceptable solution because of the constraint to get to work dressed and dry. Let us see how a synthesis system might go about "knowing" this.

As stated earlier, the synthesis system "knows" the effect of applying an operator to its arguments. Assume that the argument is a person. Further, assume that there is some list where certain properties about person is recorded. Therefore, when (dress) is applied to person, the property list of person is checked to ensure that it meets certain conditions (e.g., not already dressed, awake, etc.). We assume that person meets the necessary conditions for applying (dress). Then (dress) causes the property list associated with person to be updated with certain other properties (e.g., has on clothes, has on shoes, etc.). With regards to the particular person that (dress) has been applied to, (dress) changes to (DRESS) and person has the property of being dressed.

Next, the operator (shower) is applied to person (i.e., the same person to which (dress) has been applied). The operator (shower) is designed to update the property list of its argument, whereby, all objects defined by the "has-on" relation, for example, are changed to "is-dry" = false. Since a goal is to have "is-dry" = true, the synthesis system can be structured to not do actions that contradict a task it is to

achieve. So then, by using certain data structures and knowing "a priori" the affect of an operator, it is possible to determine if a certain sequence of operators produce the desired result.

Finally, when all these concerns are considered in total, we are able to determine the problem domain of the synthesis system. The problem domain represents the kinds of problems for which the synthesis system can provide solutions (programs). For example, in the above example we might consider the problem domain of the hypothetical system to cover the "morning grooming for adults". We specified adults because our system does not handle infants - it does not know how to "diaper" (dress) a baby. So in some sense, the problem domain indicates the power of the synthesis system. As the class of problems increase for which it can provide solutions, the power of the synthesis system increases.

This simplification of the program synthesis process has shown the importance of the specification to the synthesis system. In assuming the program synthesis system to be powerful enough to achieve all the subgoals belonging to some problem domain, it is clear that the specification must convey the necessary tasks and constraints. A good set of specification should be minimal and complete. That is, it has no redundant pieces of information and the addition of any information would either be redundant, or it would change the originally described product to being something else.

I. Benefits of Automatic Program Synthesis

Simon in [1] points out that our basic understanding of artificial intelligence depends on how well we can define the problem we are trying to solve. As such, automatic program synthesis provides an excellent domain for experimentation with problem representation. Simon feels that our ability to extend AI in other ill-defined areas depends on our ability to represent problems. Further, this area offers excellent opportunties for situations where information and meanings have to be communicated for a definite purpose (e.g., natural language). This capablitiy could also provide exciting new possibilities for advancing general research by providing a test bed for new ideas. The possibilities are endless. But the most exciting, I think, is having an alternative method for doing software development and software maintenance at reduced cost.

IV. STATEMENT OF THE PROBLEM

A. GENERAL

Chapters I and III of this thesis have provided us with an intuition for what the "sequence problem" is. In this chapter, we will state more precisely the "sequence problem" associated with automatic program synthesis, present the hypothesis of this thesis, and give an overview of some of the important work in automatic program synthesis.

B. THE SEQUENCE PROBLEM

Let us agree that "programming" is a transformation of ideas into some machine useable form that accomplishes a set of actions. For those of us who have done programming, several things about programming were quickly learned. First, we learned that certain tasks within the program must be accomplished before certain others if the program was to be correct. This same requirement exists within an automatic program synthesis system. Secondly, we learned that there are often many different orderings of tasks that provide a solution to the programming problem. But we also learned there are many orderings that do not provide a solution. With these ideas in mind, we can precisely state the "sequence problem". The sequence problem associated with automatic program synthesis is how to determine a correct ordering for

achieving, in the target language of the synthesis system, the subgoals that comprise the FIND portion of a specification. Obviously, if the ordering is followed we want it to provide a solution to the problem, if a solution exists. But we also would like a method that can accomplish this without searching through a combinatorial explosion of possibilities. Why? Let us say that a problem can be specified with some finite number, n, subgoals that comprise the Find portion of the specification. Since there are n! orderings, as n gets arbitrarily large we would like for our method to still be effective in obtaining a solution sequence.

C. PROBLEM STATEMENT

If we consider the nature of program synthesis as given in Chapter III, we will recall that the synthesis process begins by acting upon the specification. "One of the principal difficulties in top-down design is knowing how to decompose a problem into subproblems. At present general knowledge of this kind is intuitive and not in a form suitable for automation. Rather than attempt to formalize this general knowledge we focus on special ways to decompose a problem" [6]. The focus of this thesis is the problem of decomposing the given problem specification. Our goal is a method that considers the entire specification before providing a sequence of subspecifications. Each subspecification of the sequence

represents a subproblem, and the composition of their solutions is a solution to the given problem specification.

I. HYPOTHESIS

The hypothesis of this thesis is as follows:

> By extracting information contained within the procedural specification, semantic networks can be constructed and used to determine a sequence for achieving subgoals that, if followed, provides a solution to the procedural specification.

II. REVIEW OF RECENT WORK

An important and fundamental fact about program synthesis was described by Barstow in [9]. Barstow performed an experiment based upon the observation that human programmers know a lot about programming. He also noted that much of this knowledge seems to be independent of any particular programming language. This knowledge comprises a variety of concepts (e.g., sorting, pattern matching, sets), specific implementation techniques (e.g., hashing, binary search, quicksort), heuristics for suggesting implementation techniques, and general strategies for various situations (e.g., divide and conquer, greedy).

Barstow used this information and developed a system, called PECOS, which established a feasibility milestone for automatic program synthesis. Barstow showed that human knowledge about programming can be made precise enough that it can be codified for machine (computer) usage. PECOS resulted in a 400 rule knowledge base system that constructed (i.e., not synthesized because the algorithm was supplied) programs through a gradual refinement approach; almost all program synthesis is done by gradual refinement.

Manna and Waldinger in [10] incorporated theorem proving, unification techniques, mathematical induction, and transformation rules within a single system. This provided them with a simpler program synthesis structure. They make the important claim that program synthesis systems always require theorem provers. Also, they show that theorem provers capable of handling existsential quantifiers (e.g., there exist at least one such item that makes this true) are important to the ability to introduce recursion or iterative loops into a program's structure.

Probably the most acclaimed of all works concerning automatic program synthesis is [5] by Manna and Waldinger. Among this work's many important principles for synthesizing iterative straight-line programs, is the concept of protection for achieving simultaneous goals. Waldinger in [8] shows that protected relationships (those relationships needed to prove that some aspect of the program is true) should not be

violated (i.e., made to be false once established as true , even temporarily, because an infinite series of synthesis actions could result.

From the preceding chapters of this thesis we have gained an understanding for the important concept of sequencing as it is associated to automatic program synthesis. Although much work has been done in the area of planning we will consider Sacerdoti's work, namely that of [11] and [12]. In [11], Sacerdoti developed a system, called NOAH (Nets of Action Hierarchies), of techniques for generating a hierarchical plan capable of providing varing degrees of detail for any level within the hierarchical plan. NOAH exploits the concept of representing a plan as partially ordered sequences that postpone ordering commitments until sufficient information exists to determine the task order. Also, NOAH is capable of reviewing and improving its plan. The plan uses a structure called a procedural net. The net is a graph structure whose nodes represent actions at varying levels of details, organized into a hierarchy of partially ordered time sequences. A significant contribution of Sacerdoti's work is viewing information at appropriate levels of detail for the purpose of determining a sequence for achieving tasks.

In [12], he applies NOAH to the Blocks World problem domain and solves Sussman's Anomaly, a classic nonlinear problem in automatic program synthesis. The nonlinear nature

of this problem will be explained in Chapter VI, where we present the problem in detail.

Sussman in [13] presents a program called HACKER that is capable of doing limited program synthesis. HACKER also displays the ability to learn from previous mistakes by viewing debugging as a positive set of circumstances. That is, when trying to extend code for some new situation, the reason for the failure is located and the old code is extended as a result of locating the reason for failure. "The old code serves as a 'plan' for the new code"[13]. The HACKER system simulates a robot and a table with blocks on it; the robot moves the blocks according to some set of rules for the purpose of achieving some specified goal state.

Today, however, only small and rather simple problems can be synthesized. Many experts acknowledge the difficulty associated with synthesizing large problems like operating systems but feel that someday this kind of expertise in Artificial Intelligence will be in place. In fact, "many of the experts believe that the artificial intelligence of machines will one day surpass the natural intelligence of man . . [2]." On the other hand, many experts express their disappointment about what Artificial Intelligence has actually delivered when compared to its initial claims and the expectations of those anxiously waiting for AI products. As such, we see that the opinions vary greatly about what is or is not possible within and by AI, a reason for continued

research. But certainly all seem to agree that the gap between the current technology and our desires needs to somehow be bridged.

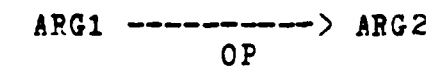
V. METHODS

A. GENERAL

The following methods will be used to extract information from the procedural specification, construct a graph-like structure called a semantic network, impose levels onto the semantic network, and derive a sequence of specifications that satisfies the original procedural specification, if such a sequence exists. Nilsson in [14] defines a semantic network (henceforth called a semantic net) as a collection of predicate calculus expressions represented by a graph structure. For the purpose of this thesis, the predicate calculus expressions that constitute the semantic net will be the relationships contained in the procedural specification.

We will assume that the procedural specification consist of unary and binary relationships. Thus, the relational operators can be represented by labeled arcs; the tail of the arc leaves the node representing the first argument, and the head of the arc enters the node representing the second argument. For example, if OP is a relational operator and ARG1 and ARG2 are arguments, then a subgoal of the procedural specification would be denoted as (OP ARG1 ARG2). In the case of a unary relationship, a dummy argument is used as the first argument, namely DUMMY. By introducing the dummy argument for unary relationships all relationships share the same format.

Thus the expression (OP ARG1 ARG2) is represented by the following structure:



The following abstractions are imposed onto the semantic net. First, we impose on it the concept of level that is associated with tree type structures. We also impose the idea that an arc's level designation is determined by the level on which its entering (terminal) node resides. For example, if a terminal node resides on level 1 of the semantic net, then all arcs entering that node are designated as belonging to level 1 of the semantic net. The DUMMY node always exists in the semantic net and is at level zero. An example of a semantic net with levels, the motivation for the idea of levels, and a definition for computing the level of a node are presented in the next of this thesis.

3. AN EXAMPLE PROBLEM

For an example problem, suppose we are given the following procedural specification:

```
[(WHERE (OP1 A B)(OP2 B C)(OP3 B D)(OP4 D C))
 (FINI (OP5 C B)(OP6 C A)(OP7 DUMMY D)(OP8 B D)(OP9 A D))]
```

Also, for any program synthesis system, the number of relational operators that are available for specifying the

specification is finite; the program synthesis system is finite. Note that CP7 is a unary relational operator and different from the other operators because they are binary.

Recall that the WHERE portion of the procedural specification refers to the current start state and the FIND portion refers to the goal state. A semantic net is constructed for each, which we call the Where-net and Find-net, respectively.

1. Net Construction

Let us begin by constructing the Where-net; the Find-net is similarly constructed. We arbitrarily get a subgoal and transform it into its representative nodes and arc form, which we will refer to as a "labeled arc". We refer to it as a "labeled arc" because we want to be able to identify and associate each arc of the net with the subgoal it represents in the specification. As such, a unique value is assigned to the subgoal, and the same value is assigned to its corresponding arc. We adopt the scheme of assigning the positive integer i to the i-th subgoal placed into the semantic net. This same notion can be extended if multiple arcs are created by a subgoal.

We get the first subgoal, convert it to its arc/node form and label its arc with the value 1. We then place the non-terminal (source) node on level 0 (zero) and the terminal node on level 1 of the semantic net. We then connect these nodes with an arc and assign the arc its unique value. The

next subgoal is selected and transformed into its arc/node structure in the same way. If neither of its nodes exist in the semantic net, then it too is placed into the net in the same manner as the first subgoal. Otherwise, one or both nodes exist in the net and the following is done.

Let us consider the case where only one node of the current labeled arc exists in the net. If the source node for the current subgoal we are attempting to connect exists in the net, then we place the terminal node on the next higher level and connect them with the current arc, giving it its unique arc value. If the terminal node exists and is not on level $\emptyset$, then the terminal is on level i and we place the source node on level i-1 and make the connection. Otherwise, the terminal node is at level $\emptyset$. In this case, the terminal node and all of its descendant nodes are placed on the next higher levels (i.e., increase each of these nodes' level designation). The source node is placed on level $\emptyset$ and connected to its terminal node by its labeled arc.

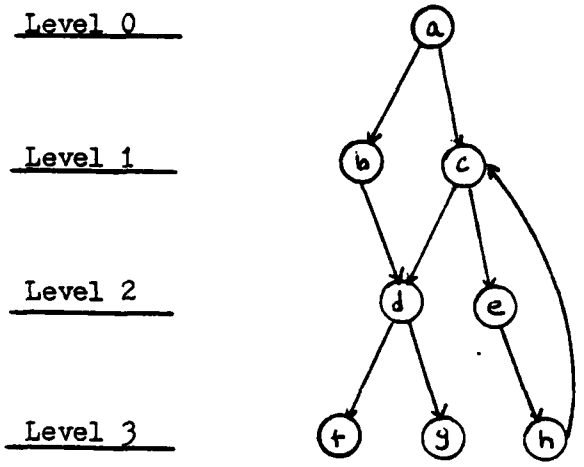
When both nodes of the current subgoal already exist in the net there is the potential for causing a loop or cycle in the net. We first consider the case where no loop is formed in the semantic net (i.e., the terminal node is not on the path from the root to the source node). When the nodes of the current subgoal exist in the net and the terminal node is on a higher level, then the arc is connected and labeled. Otherwise, the terminal node is on the same or lower level. In

this case, the terminal node and all of its descendents are placed on the next higher level and the connection is made accordingly. Because we impose the restriction that a terminal node must reside on a level greater than any of its source nodes, loops must be handled in a different manner.

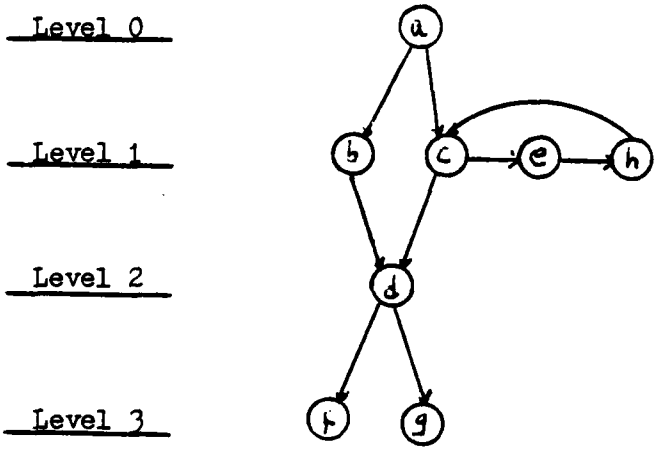
For the case where an arc/node structure causes a loop in the semantic net, the loop is "localized" to reside on the highest level for which a node in the path of the loop is a terminal node, and its incoming arc is not on the path of the loop. By localizing, we mean placing all the nodes in the loop, and their respective labeled arcs, on the above determined level of the semantic net. The descendant nodes of the nodes in the loop are likewise adjusted to reside on their appropriate level. If there is no incoming arc into the loop, then the localized loop will reside on level 1 of the semantic net. Figure 5.1 shows an example of localizing a loop.

As a result of localizing a loop, the set of nodes and arcs that form the loop are considered as a single entity. That is, if one node is caused to reside on some higher level, then all other nodes will reside on the new level. Descendant nodes are adjusted to appropriate levels based on the conditions caused by localizing.

Also, arc/node structures that are duplicate are not allowed in the net and are considered to be redundant information. A duplicate arc/node structure is one with the



(a) Unlocalized



(b) Localized

Figure 5.1
Localizing a Loop

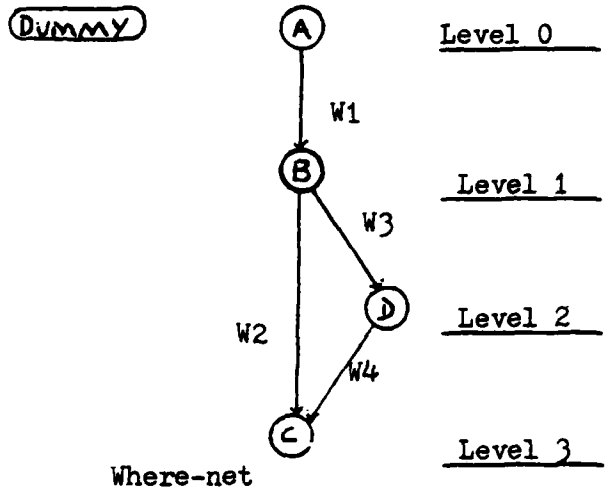
same terminal and source nodes and the same binary operator and that already exists in the semantic net.

Once the semantic net has been constructed arcs are considered to be undirected. Directed arcs help to stabilize the structure by providing the organization necessary to partition subgoals into a partial ordering. The directed arc causes the task to be pushed into an "execution time frame." For example, s9 and s3 are subgoals that reside on level 9 and level 3 of the same semantic net, respectively. If some sj is introduced in the net that has as its source node the terminal node of s9 and has as its terminal node the source node of s3, then the subgoal s3 would be pushed to level 10 of the semantic net. The motivation for this idea is given below when we interpret the semantic net. Figure 5.2 shows the Where-net and Find-net for our example.

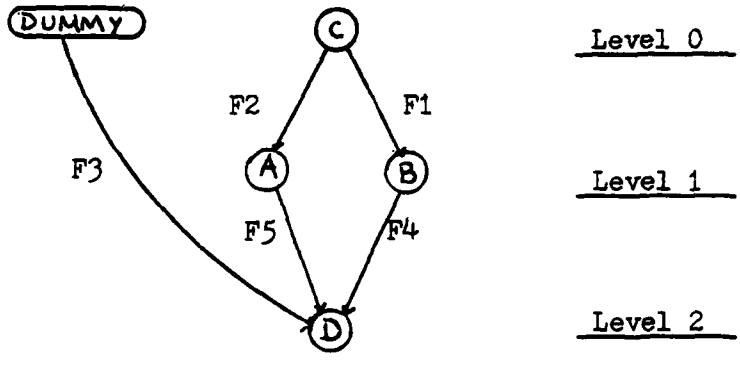
2. Interpreting the Semantic Net

After constructing the Where-net and the Find-net, there is enough information to determine the sequence for accomplishing the subgoals. At this point, most of the emphasis is placed on the Find-net. The Where-net is used to establish existing relationships of the current world view, and is updated to reflect the accomplishments gained in determining the order for a subgoal. However, the reasoning language will use both nets to deduce knowledge.

This knowledge represents the manner in which the updating of the Where-net with a subgoal from the Find-net



Where-net



Find-net

Figure 5.2
Where-net and Find-net for Example Problem

affects the ability to update the where-net with the remaining subgoals of the Find-net. The essence of this knowledge is to determine if prerequisite conditions are needed so that a subgoal from the Find-net can be included into the where-net, and/or determine if subgoals are permissible under the rules that define the problem domain. These features of the reasoning language will be shown in the presentation of specific examples of the next chapter. This brings us to the two aspects of the Find-net that need interpreting, which are: the levels of the net, and the arcs that span the same level of the net.

a. Interpreting the Levels of the Net

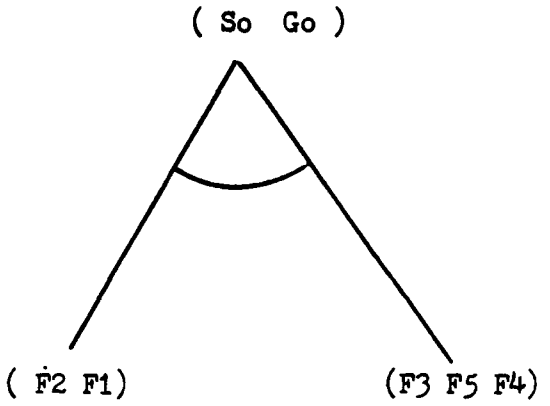
The general interpretation given to levels is that subgoals on a lower level (i.e., closer to the root) are required to be achieved before achieving those subgoals of a higher level. This interpretation assumes that the intermediate argument "b" of the transitive property will somehow be modified if aRb is achieved. Recall that aRb equates itself to an arc/node structure of the semantic net which itself equates to a subgoal of the procedural specification. This approach is conservative from the view that if the subgoal defined by aRb modifies its argument b, then the subgoal defined by bRc will have already inherited the modification. Specifically, however, the Find-net in Figure 5.2 shows that subgoal 1 and subgoal 2, denoted by F1 and F2, are to be achieved before all others. Then, in some

order to be determined later, F4, F3 and F5 are to be achieved. In short, we have determined which subgoals belong to "level 1 tasking" of the original problem.

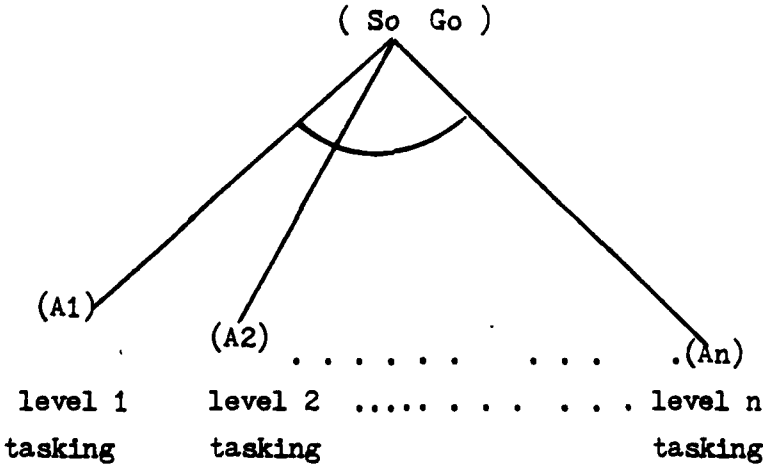
The graphical representation of this information is the AND/OR graph shown in Figure 5.3(a) below. Figure 5.3(b) is the general form of the AND/OR graph produced by the methods described in this thesis. The interpretation given to this graph is that each leaf node represents a subgoal for the synthesis system. We can also see that a leaf node can be composed of several subgoals from the FIND portion of the procedural specification. The root node of the AND/OR graph represents the procedural specification. The sequence we have sought after is the left-to-right ordering of the leaf nodes in the AND/OR graph. If we use the graph of 5.3(b), then the sequence is achieve A1 and then achieve A2 and then achieve A3 etc.

b. Interpreting Subgoals Spanning the Same Level

To sequence subgoals on the same level of a semantic net requires a large amount of knowledge about the problem domain. For example, such a capability is dependent upon "knowing" the meanings of all the possible connections that are allowed on a level within a semantic net. The synthesis system must be able to "know" how these connections impact the solution. For example, suppose we are given two square blocks, block a and block b. Further, suppose we are



(a) Example Problem



(b) General Problem

Figure 5.3
AND/OR Graph

given one relation which states that block b is on the top of block a. Also, suppose another relation, belonging to the same procedural specification, states that block a is on the top of block b. When we consider each relation (subgoal) separately it represents a valid condition, and one that is possible to achieve. However, when we consider these subgoals together they represent an invalid condition (i.e., only one block can be on top) and the reasoning language must be able to "know" this.

Additionally, the number of nodes residing on a level represent the number of parallel processes that are possible during the processing of a particular level. Each process consists of or is defined by the subgoals (i.e., arc/node structures) for which the node in question is the terminal node. Within Chapter VI we will see how knowledge specific to a problem domain is applied to subgoals of the same level. Figure 5.4 shows all the possible isomorphic connections that are possible in a semantic net when using only two subgoals. There specific interpretations are dependent upon the task-specific knowledge called critics that are associated with a particular problem domain. We will consider what these networks mean within a specific problem domain in the next chapter of this thesis.

In general, however, critics are applicable for use with the reasoning language or target language. These critics are the conditions or circumstances for applying or

not applying a certain sequence of operators. Simply, critics provide the strategy or control structure for the detail aspects of some problem domain. They are the "fine tuning" mechanism for the overall or more general control strategy chosen to perform automatic program synthesis synthesis. In [5], Manna and Waldinger refer to critics as strategic conditions. Sacerdoti [10 and 11] and Sussman [12] refer to such an approach as the criticism approach.

An obvious concern at this point is how do we consider loops. A loop is considered to be a group of subgoals coupled sufficiently tight that this method is unable to sequence.

c. Isolating Information Relevant To a Subgoal

Information relevant to any subgoal contained within the specification is easily obtained. By relevant we mean possibly impacting the result or ability to achieve some condition. Again, observe Figure 5.2 above. To obtain all relevant information about a subgoal, say, F4 for instance, its arc is located by using the enumeration value assigned to the subgoal. The terminal nodes of the subgoal are then located in the semantic net. From those applicable nodes, the list of values of arcs along the path(s) back to the root of the net represent all the information relevant to that subgoal. In this particular case we see that all the other subgoals are relevant to subgoal F4.

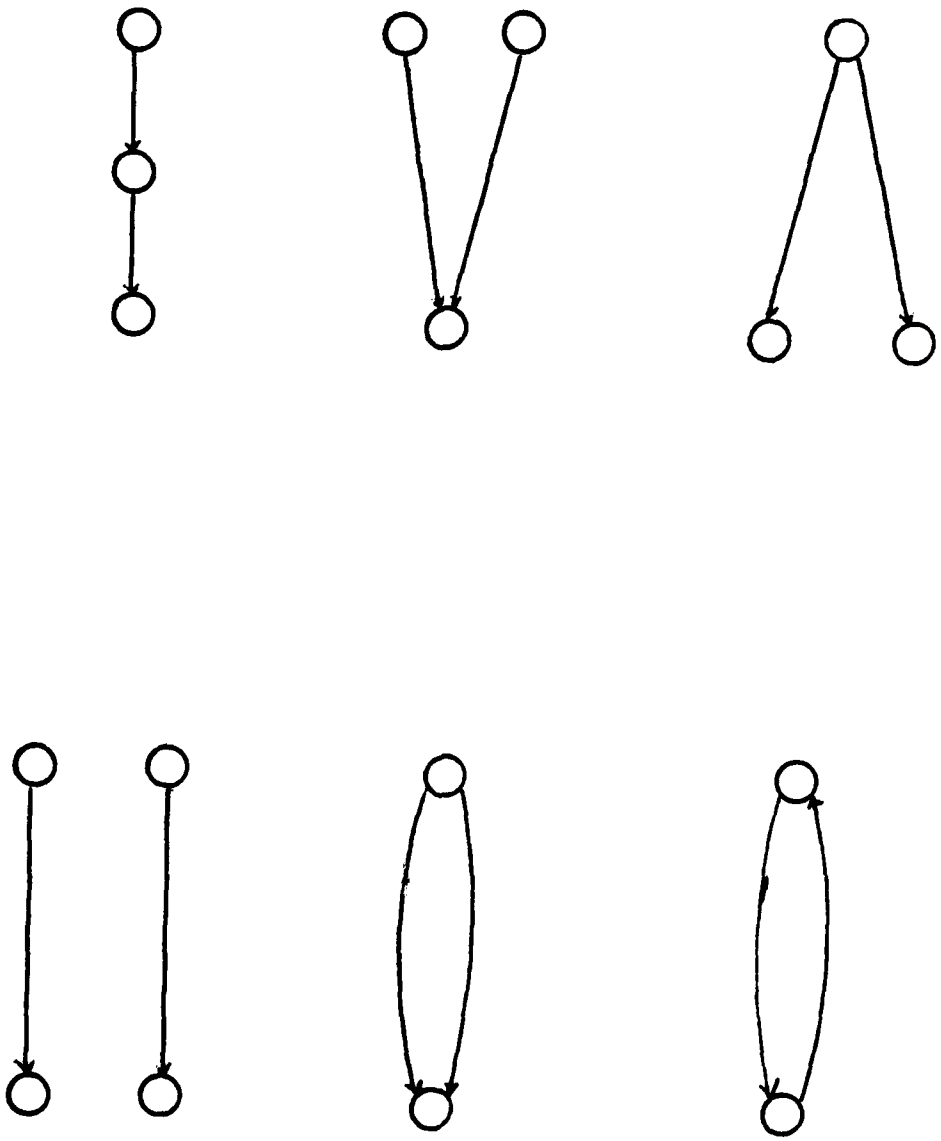


Figure 5.4
Isomorphic Nets Using Two Subgoals

C. RATIONALE

Let us consider the above interpretations. The overall scheme is based upon the notion referred to herein as the "transitive interplay of subgoals". When we examine the criterion of achieving subgoals belonging to level i before those of level i+1, we imply that the subgoals of level i impact the achievement of the overall goal before those goals belonging to level i+1. Therefore, there is the requirement to put in place, first, those foundational "things" of level i.

What are those "things"? They are relationships. The semantic net is constructed from relationships that comprise the procedural specification. As we focus upon the Find-net, we see that it does more than state relationships which represent a problem solution. It also implies the usage of a certain set of operators. That is, to achieve the relationship only certain operators are applicable because operators are limited by the arguments they are defined over and their resulting action - making true certain protected relationships.

That information is then incorporated into the reasoning language of the synthesis system in the following manner. Since arcs correspond to relations and the nodes correspond to arguments, the attributes or properties of arguments are "reasoned" in the reasoning language to determine if the prerequiste condition(s) exist(s) or are needed in achieving a subgoal. Likewise, when multiple relations affect a node the

same strategy exists. The only difference in this case is the set of applicable operators may have increased in number.

By imposing the notion of direction or flow onto the binary relational operators, we can create mappings as

$$OP(X) \text{ --------} > Y$$

just as $f(x)$ ------> y is normally considered in functional notation. We then connect these mappings, and the resulting semantic network shows the transitive affect of operators upon arguments. To explain what we mean by the term "transitive affect", we refer to the definition of the transitive property of a binary relation. Formally, the transitive property is defined as: if aRb and bRc then aRc, where R is some relation over A x B, and a and b of aRb are elements of the set of objects A and B, respectively. We extend R to be some family or set of relations over a problem domain for which the synthesis system has been designed. This extension increases the arbitrariness of the relations R that compose the formal definition of transitivity. That is, if aRb and bRc then aRc means each "R" of the transitive definition may in fact be different from each other.

Furthermore, if we transform the antecedent of the transitive definition (i.e., the If-Clause) into mappings and construct a network we obtain the following:

P(a) -------> b and R(b) ------> c

a ----------> b ----------> c
R1 R2

where R1 and R2 are members of the set R of relations for the problem domain. The interpretation given to the consequence of the definition aRc is: the relation aRb is necessary to the relation bRc because the reasoning language assumes that it possible for the target language to modify the intermediate argument "b" as a result of achieving aRc. Stated in a different way, through "b" the relation bRc inherits any possible changes that previous operations might have caused to occur to the argument "b".

Let us consider the following. First, subgoals direct the synthesis system to accomplish some task. In so doing, it produces associated blocks of code, represented as cI, which has its associated protected relationships, denoted as rI. If, we assume that the synthesis system is able to process the entire set S of subgoals that comprise the procedural specification, then a program P is derived.

We assume that program P satisfies S. Therefore, P is made up of the sequence of blocks of code C for which no contradiction or violations exist.

Consider a procedural specification where two subgoals, sI and sJ, comprise the Find-net. When each subgoal is accomplished, its accompanying protected relationships are

also established, namely rI and rJ. Since relationships are statements about arguments, a violation can occur only if sI and sJ contained a mutual argument, implied or explicitly stated. Similarly, if an argument is only contained within one subgoal, then no circumstance for contradiction exist.

I. ALGORITHM

The following algorithm is used to describe more precisely the methods for determining the sequence for achieving the subgoals contained in the procedural specification. The algorithm assumes that subgoals are represented as binary relations. Acronyms, variable names, data structures and the like will be as consistent as possible to previously used names and terms within this thesis.

We will show in Chapter VI how task-specific knowledge is incorporated into the structure of SEMANTIC-SIG. Again, our aim is not to present a fully detailed reasoning system. We are concerned with the treatment of the entire problem specification and the usefulness of the semantic net in determining a sequence of subspecifications.

SEMANTIC-SEQ is the global control procedure for finding our desired sequence. Its input is the procedural specification of the original problem and name of the problem domain, and its output is a sequence of specifications , whereby the sequential achievement of the first, second, etc. specification, in that order, lead to a solution. The semantic nets (i.e., Where-net and Find-net) are represented as adjacency lists.

```
Algorithm SEMANTIC-SEQ ( specification, proo-domain )
begin
1.    Where-por <--- WHERE portion of specificatior;
2.    Find-por <--- FIND portion of specification;
3.    BUILD-SNET(Where-net, Where-por);
4.    BUILD-SNET(Find-net, Find-por);
5.    REASON(Where-net, Find-net, prob-domain);
6.    WRITE-SPEC(Where-por, Find-por, Find-net);
end SEMANTIC-SEQ ( specification )
```

BUILD-SNET constructs the appropriate semantic net from the accompaning constraint portion (the WHERE portion or the FIND portion) of the procedural specification. SNET is the constructed semantic net that is produced by BUILD-SNET. The semantic net is constructed by placing the binary relations in the adjacency list using CONNECT.

```
        Procedure BUILD-SNET ( SNet, constraint )
        begin
 1.        SNet  <---  null;
 2.        Open-list  <---  constraint;
 3.        while  Open-list not empty do;
            begin
 4.            binary-rel  <---  first conjunct of Open-list;
 5.            delete first conjunct from Open-list;
 6.            CONNECT(SNet, binary-rel);
 7.       return;
            end
        end BUILD-SNET ( SNet )
```

CONNECT converts the binary relation into its arc/node form and places it into the adjacency list. Lines 4 through 10 are concerned if both nodes already exist in the net. If so, and the binary operator is already in the label function, LABEL, for that pair of nodes, then a duplicate arc exist. Lines 7 and 8 determine if this arc will create a loop, if so, an error condition.

FIND(SNet,node) is a boolean function that returns TRUE if node is on the adjacency list SNet. Otherwise, it returns FALSE.

LEVELNUM(node) returns the level that a node resides on in the semantic net.

LEVELSHIFT(SNet(node),level-designation) moves all nodes on a list that have the same level designation to an appropriate level (i.e., higher than any of its source nodes) and performs the necessary housekeeping like updating the LABEL and LEVEL functions.

LABEL(source,terminal) is a function that labels the edge between the nodes source and terminal (i.e., the relational operator).

LEVEL(level-designation) contains all the binary relations (subgoals) that reside on level "level-designation" of the semantic net.

```
        Procedure CONNECT ( SNet, binary-rel )
        begin
 1.       bin-op  <---  binary operator of binary-rel;
 2.       source  <---  arg1 of binary-rel;
 3.       terminal  <---  arg2 of binary-rel;
 4.       if FIND(SNet, source) and FIND(SNet, terminal) then
            begin
 5.           if LABEL(source,terminal) equal bin-op then
 6.              ERROR-RETURN;
 7.           if terminal on the path from root
                to source   then
 8.              LOCALIZE(SNet,binary-rel);
 9.           if LEVELNUM(source) not less than
                LEVELNUM(terminal) then
10.             LEVELSHIFT(SNet(terminal),LEVELNUM(source));
            end
11        else-if FIND(SNet,source) then
12.        LEVELNUM(terminal)  <---  LEVELNUM(source) + 1
13.     else-if FIND(SNet,terminal) then
14.        if LEVELNUM(terminal) not equal zero then
15.           LEVELNUM(source)  <--- LEVELNUM(terminal) - 1
           else
             begin
16.           LEVELNUM(source)  <---  0 ;
17.           LEVELSHIFT(SNet(terminal),0);
             end
```

```
          else
            begin
18.          LEVELNUM(source)  <---  2 ;
19.          LEVELNUM(terminal)  <---  1 ;
            end;
20.       add terminal to SNet(source);
21.       add the list "headed" by terminal to SNet
             such that SNet(terminal) exists in SNet;
22.       add bin-op to LABEL(source,terminal);
23.       add binary-rel to Level(LEVELNUM(terminal));
24.       return;
        end CONNECT ( SNet )
```

REASON(SNet1,Level-SNet2,Prob-domain) achieves the subgoals contained in the Level structure of SNet2 (i.e., the Find-net) by incorporating them in SNet1. SNet1 represents the start state. Line 7 determines the manner in which the subgoals will be retrieved from the Find structure. Such a manner is problem domain dependent. Line 9 of REASON determines if the subgoal causes a violation by reviewing the effect of the binary relation with the particualr "critics gallery" defined for some some problem domain.

When the Hold-List is empty, all subgoals have been successfully placed or achieved in the SNet1. Line 14 determines if any prerequisites can be achieved from any of the subgoals that were prevented from being achieved in SNet1.

Not being able to establish any prerequisites means that the synthesis system is not able to continue the process (i.e., those subgoals on the Hold-List can not be incorporated into the SNet1). Lines 15 and 16 show how the FIND portion of the procedural specification is modified with prerequisites.

```
        Procedure REASON ( SNet1, Level-SNet2, Prob-domain  )
        begin
 1.      LOOP  <---  TRUE;
 2.      while LOOP do
          begin
 3.        SimNet  <---  SNet1;
 4.       SimLevel  <---  Level-SNet2;
 5.       Hold-List  <--- empty;
 6.       while SimLevel not empty do
            begin
 7.           binary-rel <--- CRITIC-2(SimLevel,Prob-domain);
 8.           delete binary-rel from start of SimLevel;
 9.           if CONFLICTS(binary-rel, Prob-domain) then
                 add binary-rel to Hold-list and
                 add established prerequisites of
                 binary-rel to Invariant-list
              else
11.              update SimNet with binary-rel;
            end;
12.       if Hold-List is empty then
```

```
13.          return;
14.        if GENERATE(prerequisite,Hold-List) then
             begin
15.          add prerequisite and Invariant-list to
             appropriate level of Level-SNet2;
16.          add prerequisite and Invariant-list
             to SNet2;
             end
           else
17.          LOOP  <--- FALSE;
           end
18.      OUTPUT no solution, current value of Hold-List and
19.      ERROR-RETURN;
         end REASON ( SNet2, Level-SNet2 )
```

WRITE-SPEC is the output function of SEMANTIC-SEC. The where-por represents the WHERE portion of the original procedural specification. Level-SNet2 is the goal state that has been possibly modified by adding "prerequisite subgoals" and ordered in a sequence that if solved by the synthesis system a solution is provided.

```
        Procedure WRITE-SPEC ( where-por, find-por,
                                          Level-SNet2 )
        begin
1.      I  <---  starting level value for Find-net
                 (e.g., top or bottom level);
2.      start-goal-list  <---  where-por;
3.      next-goal-list  <---  null;
4.      orig-goal-list  <---  find-por;
5.      Working-SNet2  <--- Level-SNet2;
6.      while Working-SNet2 not empty do
          begin
7.          add all binary-rel of level I of
              Working-SNet2 to next-goal-list;
8.          delete level I from Working-SNet2;
9.          output '[(WHERE',start-goal-list,')(FIND',
                                 next-goal-list,')]';
10.         update value of I;
11.         start-goal-list  <---  next-goal-list;
          end;
12.     output '[(WHERE',start-goal-list,')(FIND',
                                 orig-goal-list,')]';
        end WRITE-SPEC ( sequence of specification, and
                                       Level-SNet2 )
```

VI. APPLICATION OF METHODS IN THE BLOCKS WORLD

A. GENERAL

The methods developed in Chapter V for determining a sequence from the subgoals of a procedural specification will be applied to the problem domain called the Blocks World. Two examples are presented. The first is a classic one that has come to be known as Sussman's Anomaly [13]. It represents the utility of these methods with nonlinear kinds of problems, the achieving of one subgoal inhibits the achieving of another even though a solution still exist to the problem. The second example is also of Sussman's. It shows the ease with which the prerequisites can be incorporated into these methods in order to expand the problem domain (i.e., increase the power of the synthesis system).

We proceed in this chapter to describe the Blocks World, state its rules, and give the semantics of some Blocks World operators. We then follow with some task-specific knowledge for the Blocks World which allows us to then make specific interpretations about possible connections between arc/node structures in the semantic net as it pertains to the Blocks World. These things having been presented, the methods are then applied to two examples.

Importantly, however. the aim is not to develop or present a complete reasoning language for the Blocks World; that misses the point. Rather the goal is to show that it is possible to consider the procedural specification "in toto", and sequence all the "pieces" of the specification in such a way that these sequenced "pieces", if solved, provide a solution to the problem.

B. THE BLOCKS WORLD

The scenario of the Blocks World environment is a table with blocks on it and a robot that moves the blocks. The problem domain is characterized as leaving the problem "reasonably" complete. That is, the Find or goal portion of the procedural specification is only concerned with a part of the universe of discourse; the other blocks are left to the discretion of the robot. Under such circumstances, the robot leaves all unspecified blocks on the table.

1. Rules of the Blocks World

The rules of the Blocks World are as follows:

- operators for moving objects allow only one object to be moved at a time thus it must not have anything on top of it.

- no two objects can occupy the same space during the same time.

- if either of the two above rules is violated an error results.

2. Semantics of the Blocks World

The meanings of the relational operators of the reasoning language and the operators of the target language of the synthesis system for the Blocks World are as follows:

operator	meaning
is-on	Given that two blocks exist, (is-on a b) is true if block a is upon block b. Otherwise (is-on a b) is false. (is-on a a) is vacuously true for any block a.
is-clear	(is-clear a) is true if and only if (is-on a a) is the only true is-on relationship. To keep the binary format, (is-clear DUMMY a) is equivalent to (is-clear a).

put-on	(put-on a b) causes block a to be placed upon block b. Implies that (is-on a b) is then true.
clear	(clear a) remove all other blocks from the top of block a. Implies that (is-on a a) is the only is-on relationship true for block a.

When we speak of the semantics of an operator we are not only addressing the actions performed but also we are addressing the relationships that can be known or derived if the operator is applied to some environment.

C. TASK-SPECIFIC KNOWLEDGE

As stated earlier any problem domain will require its own task-specific knowledge. These pieces of knowledge are called critics and are used in solving particular instances from the problem space of the chosen domain. Remember, our concern is not the synthesis of machine code for some target language. We are concerned with instances of reasoning from the reasoning language. That is, what are the domain particular kinds of information we need to decompose problems into subproblems? Hence, the critics herein refer to the reasoning language of the Blocks World. The desire is to employ those critics that have the widest scope of applicability over the domain; we

want to keep the critics gallery as small as possible for obvious reasons (e.g., search).

The critics suggested for use with these methods are simple and general in nature. They are:

> Critic-1 - no relation (subgoal) from the Find-net is allowed to be included in the Where-net that hinders or violates the inclusion of some other relation of the Find-net.
>
> Critic-2 - sequence plans begin from the bottom of the tree (i.e., from the largest level number).
>
> Critic-3 - establish as many prerequisites of the Find-net as possible without impeding any subgoal.

Recall the procedure REASON of SEMANTIC-SEQ. In particular, lines 7, 9 and 14 of REASON represent Critic-2, Critic-1 and Critic-3, respectively. Line 7 results in starting the reasoning process from the bottom of the net by extracting subgoals from the level list having the largest level designation.

Line 9 results in the following. If the current subgoal violates a prerequisite needed to achieve the Find-net, then

that action is not allowed. Note that CONFLICTS could contain isolated procedures for many problem domains. We assume that each operator in the Blocks World has created for it an add, delete and precondition lists of [15]. That is, when an operator of the reasoning language is applied it deletes all relations on its delete list from the global protect list that represents the current state, and adds all relations on its add list to the protect list. Therefore, we might have the simple structure:

```
CONFLICTS(binary-rel, Critics-gallery)
begin
  Case (Critics-gallery) of
       -
  (Blocks-World):
       If element of Delete-list for binary-rel
          match element of Protect-list
          Then
              return(CONFLICTS <--- TRUE)
          Else
              return(CONFLICTS <--- FALSE);
       -
       end-Blocks-World;
  (other domain):
       -
end CONFLICTS
```

Line 14 of REASON represents the effect of employing Critic-3. Some violation was possible which caused a subgoal to be placed on the Hold-List. Other critics have been unable to achieve the final goal state (i.e., incorporate all the subgoals of the Find-net without an intervention of a critic). Therefore, the requirement is to determine a prerequisite that is needed to estatlish the goal state. Therefore, GENERATE is as follows:

```
GENERATE(prereq,Hold-List)
begin
  prereq  <--- null;
  prereq-list <--- null;
  invariant-list <--- null;
  pointer <--- start of Hold-list;
  while (not end-of-Hold-list  AND prereq equal null) do
    binary-rel <--- next subgoal on Hold-list;
    update pointer to next goal;
    bin-op  <--- op of binary-rel
    arg1  <---  first argument of binary-rel;
    arg2  <---  second argument of binary-rel;
    Case  bin-op of
      (is-clear):
         If find (clear arg2) on where-list level(1)
          then
             add (is-clear arg2) to invariant-list
```

```
      else
         begin
         prereq <--- (clear arg2);
         add prereq to prereq-list;
         end;
(is-on):
   If find (is-on arg1 arg2) on the where-level list
      Level(Levelnum(arg2))
    then
      add (is-on arg1 arg2) to invariant-list
    else
      begin
      If find (clear arg1) and (clear arg2) on
         where-level list Level(1)
       then
         begin
         add (clear arg2) and (clear arg1) to
         invariant-list;
         prereq <--- (put-on arg1 arg2);
         add prereq to prereq-list;
         end
      Else-If find (clear arg2) on where-level
         list Level(1)
       then
         begin
         add (clear arg2) to invariant-list;
```

```
                prereq <--- (clear arg1);
                add prereq to prereq-list;
                end
            Else-If find (clear arg1) on where-level
                list Level(1)
             then
                begin
                add (clear arg2) to invariant-list;
                prereq <--- (clear arg2);
                add prereq to prereq-list;
                end
            Else
                prereq <--- (clear arg1) and
                            (clear arg2)
        end If-then-else-if;
        add (put-on arg1 arg2) to prereq-list;
    end-is-on
        -
      (other reasoning operators)
        -
end-Case
If prereq equal null
  then
    return (GENERATE <--- FAISE)
  else
    return (GENERATE <--- TRUE);
```

end GENERATE

A point to be gained from GENERATE is that other reasoning operators for the Blocks World, as well as other domains, can be easily added as the scope of the problem domain or synthesis system increases. Additionally, and as well as being a fine point, when a prerequisite is generated it is generated in terms of the target language. In this case and from the perspective of the reasoning language, the generated prerequisite is a condition that must be accomplished. Therefore, the reasoning system issues a command (i.e., achieve this goal: put-on, clear, etc.).

I. INTERPRETING THE SEMANTIC NETWORK

Let us consider what Figure 5.3 means in the Blocks World. Figure 6.1 shows the possible isomorphic nets with their associated Blocks World interpretation when using only two subgoals and the operator is is-on.

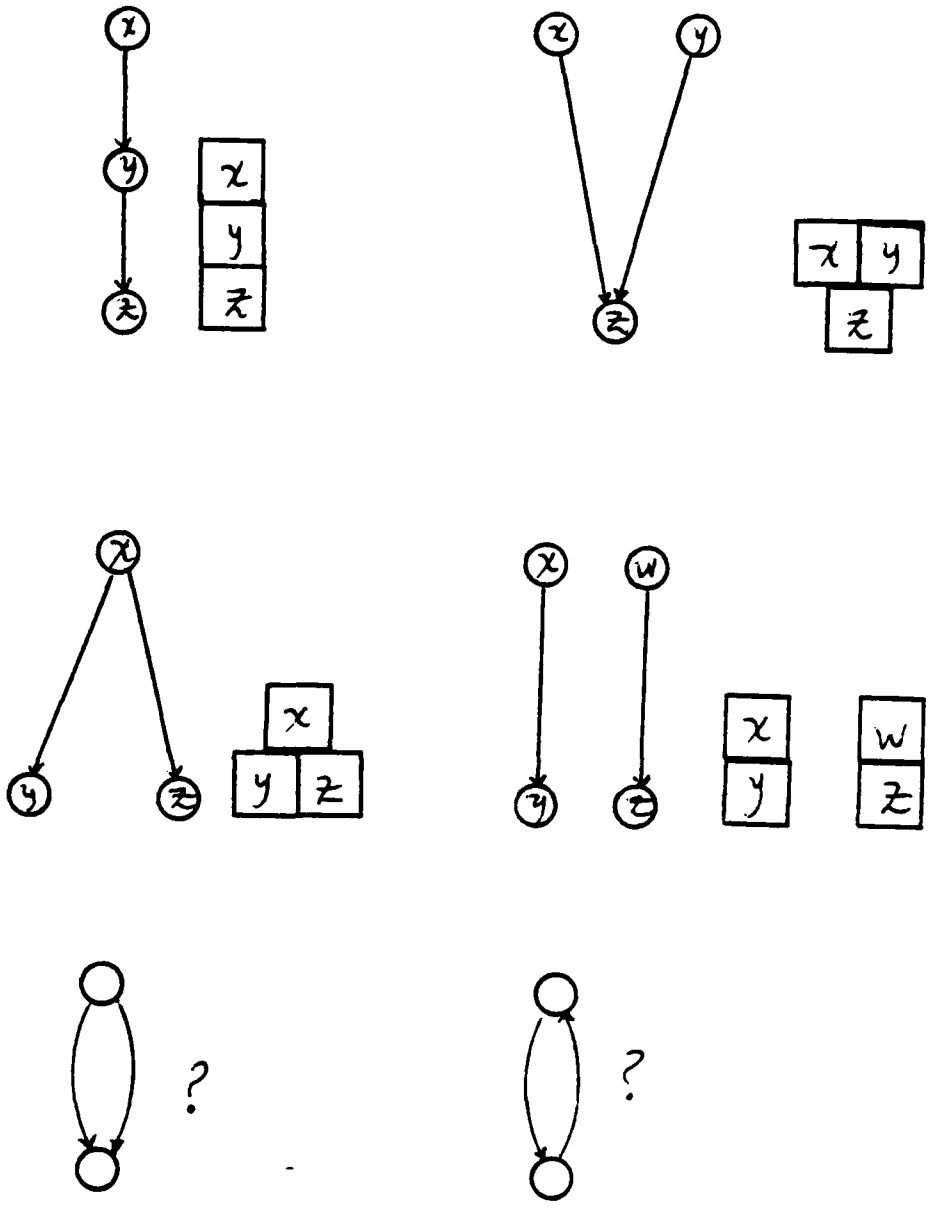


Figure 6.1
Blocks World Interpretation of Isomorphic
Nets Using Two Subgoals

E. SUSSMAN'S ANOMALY

The procedural specification of Sussman's Anomaly is:

[(WHERE (is-on c a)(is-clear c)(is-clear b))
(FINI (is-on a b)(is-on b c))]

Figure 6.2 shows the anomaly in the Blocks world format and the semantic net format.

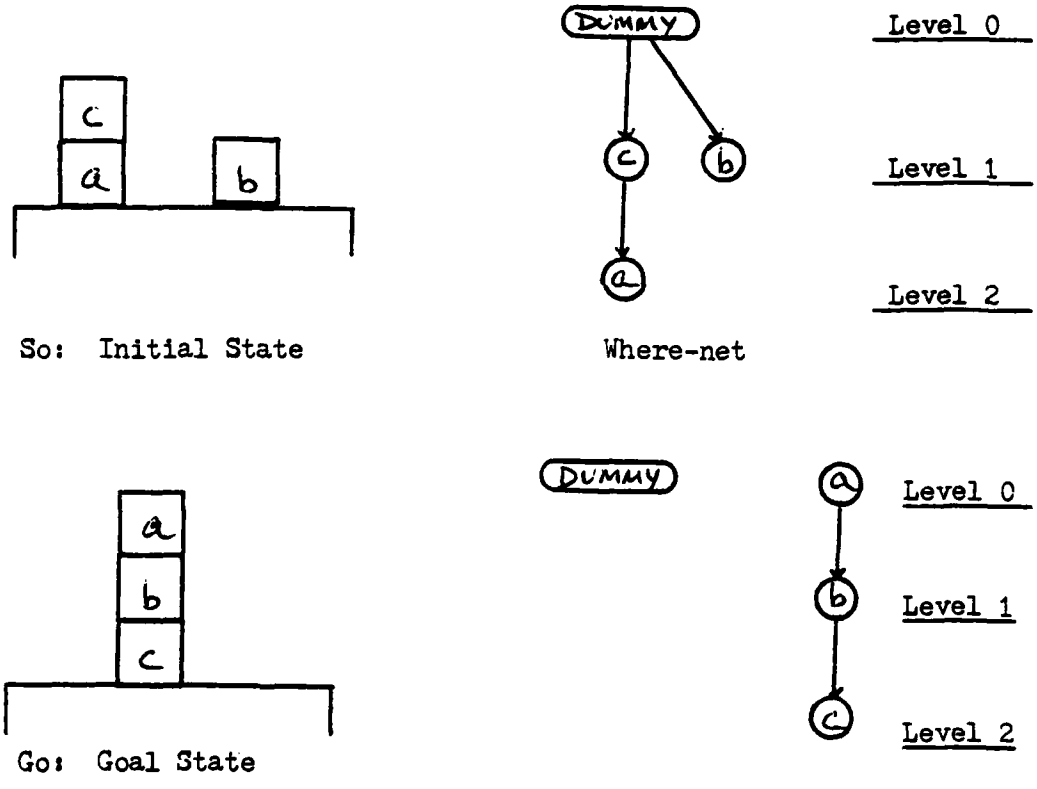


Figure 6.2
Sussman's Anomaly in Semantic Net Format

Let us consider the nonlinear nature of Sussman's Anomaly. If either conjunct (subgoal) is independently achieved then it must be undone in order to achieve the remaining subgoal. Also, as the first achieved subgoal is undone to achieve the second subgoal, an infinite loop of robot actions is created. This is the problem which Sussman's HACKER could not optimally solve [12]. Again, we can see why Waldinger in [8] insists upon protected relationships not being temporarily violated.

Two things are obvious at this point. First, the problem is solvable; we place block c on the table, then place block b on block c and then place block a on block b. Second, our critics help provide a solution. Let us see why.

First, from the procedural specification we can easily see that our methods will correctly construct the semantic nets for the anomaly. Let us see how. SEMANTIC-SEQ is invoked with the procedural specification for Sussman's Anomaly. The two constraints of the specification are somehow isolated and assigned to the variables Where-por and Find-por. Then, for each constraint, a semantic net is constructed by invoking BUILD-SNET. To construct the Where-net, BUILD-SNET is given the parameters "Where-net" and "Where-por". "Where-net" is the name that will be given to the constructed semantic net. "Where-por" is the specification to which "Where-net" is to be built. The Find-net is constructed by invoking BUILD-SNET with "Find-net" and "Find-por".

We have now reached line 3 of SEMANTIC-SEQ. BUILD-SNET is invoked. We see that BUILD-SNET begins by initializing the adjacency list "Where-net", and placing "Where-por" in a list structure. We see that BUILD-SNET takes each conjunct (subgoal) and places it in the semantic net by invoking CONNECT in line 6 of BUILD-SNET. Note that BUILD-SNET ends its execution when the list structure "Open-list" is empty.

Up to this point we have invoked SEMANTIC-SEQ with the procedural specification for Sussman's Anomaly. It assigned the two constraints to variables, and is in the process of placing the first subgoal of the Where portion of the specification into the semantic net "Where-net". Line 6 of BUILD-SNET has now executed.

CONNECT begins by decomposing the binary relation (i.e., subgoal) and assigning the its parts to variables, lines 1 through 3. Lines 4 through 10 are concerned with conditions when both arguments of the binary relation exist in the semantic net. Since this is the first connection, processing is continued. Since neither node exists in the net, the levels on which the source node and the terminal node will reside are recorded, lines 12 and 19 respectively. The actual placement of the subgoal into the semantic is accomplished in lines 20 and 21. Then the labeling of the arc is accomplished, followed by placing the entire relation on a list structure that corresponds to the level of the terminal node (i.e., the subgoal is on level 1) in lines 22 and 23, respectively. Then

in line 24 control is returned to the caller, BUILD-SNET. At this point, the first subgoal, namely (is-on c a), has been placed in the Where-net named "Where-net" at level 1.

Control is passed to BUILD-SNET. Since "Open-list" is not empty, the next subgoal is retrieved from it and then deleted from it. Control now passes back to CONNECT.

The status of "Where-net" is that it has one connection in it, namely (is-on c a). Note that DUMMY always exists in the net at level Ø. The next subgoal to be placed in the net is (is-clear c), although we use its equivalent form (is-clear DUMMY c). The subgoal is decomposed as usual but on this pass of BUILD-SNET, both nodes already exist in the semantic net. Therefore, lines 4 through 1Ø are executed.

The condition (i.e., both nodes of the current subgoal exist in the net) of line 4 is true. So then, at line 5 a check is made to see if the current subgoal is a duplicate. It is not because there is no label "DUMMY,c" in the labeling function LABEL(source,terminal). Then, going on to line 7 of BUILD-SNET we check to see if this connection will cause a loop in the semantic net. It will not, so we do not localize the loop.

Continuing with CONNECT, the condition in line 9 is tested and is true. Therefore, our first subgoal is shifted to the next level (i.e., node c is moved to level 1 and node a is moved to level 2) and the appropriate updating of data structures is accomplished to reflect the move. Then control

transfers to line 21 and the connection between DUMMY and node c is made. Figure 6.3 shows the current state of the Where-net for the Sussman's Anomaly.

Where-net

Figure 6.3
Current State of Net for Sussman's Anomaly

Since the "Open-list" of BUILD-SNET has one remaining subgoal, CONNECT is invoked again. This causes "Open-list" to become empty. Since node b is not in the semantic net, control passes to lines 11 and 12. The level designation of the terminal is placed on the next higher level from the source, since it is the source node DUMMY that exists in the net. Afterwards, the connection is made between DUMMY and node b and control then passes to BUILD-SNET. Since "Open-list" is now empty, control now goes back to SEMANTIC-SEC.

The Where-net "Where-net" has now been completely constructed. Therefore, in SEMANTIC-SEC line 4 is executed and a corresponding semantic net is constructed for the Find-net "Find-net". Figure 6.4 shows these nets.

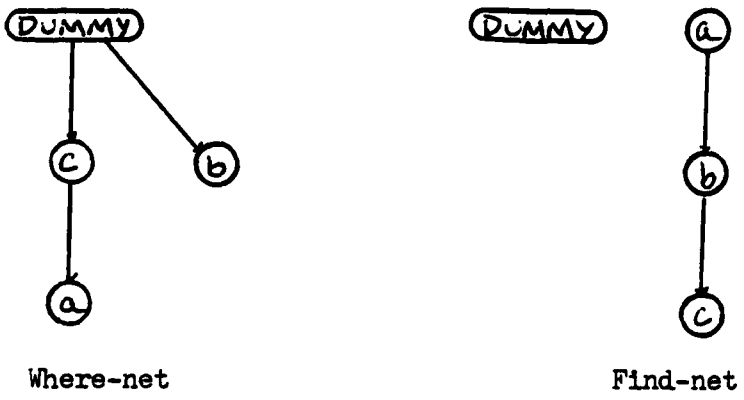


Figure 6.4
Constructed Semantic Nets

At this point in the Algorithm SEMANTIC-SEQ, both semantic nets have been constructed. The tasks left to be accomplished are to "reason" the sequence of specifications, if possible, and output them. This puts us at line 5 of SEMANTIC-SEQ. We see that the procedure REASON is invoked using the two newly constructed semantic nets, "Where-net" and "Find-net" respectively.

Since the task at hand is to "reason", let us see what pieces of information (i.e., data structures, rules, etc.) are available to perform reasoning. Figure 6.5 shows the data structures available as a result of constructing the semantic nets.

Additionally, there is the task-specific knowledge that is available that we refer to "in toto" as the Critics Gallery. As stated, the critics we present are concerned with the "reasoning" aspects of the problem. Again, the goal here is to show that semantic nets can be used to determine a sequence of specifications.

Note that our overall strategy is to start from the bottom of the net in determining our sequence. This information is provided by Critic-2. The reason for this information is simple. Since we are constructing something physical, the top of the goal state represented by the Find-net corresponds to the top of that physical structure. In our particular case, the top of the semantic net represents the top of the structure of blocks. Therefore, since the Blocks World makes

Sussman's Anomaly

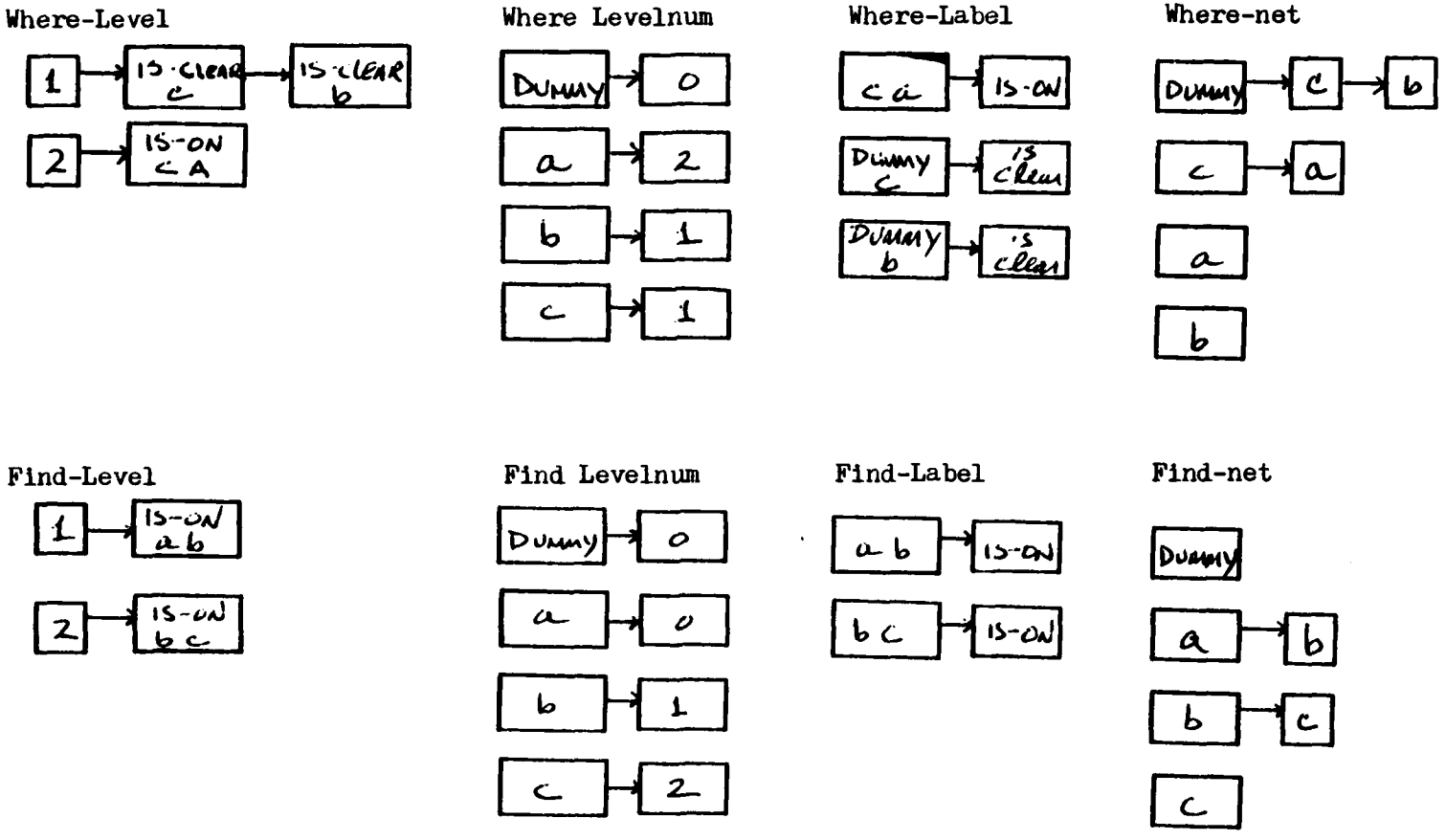


Figure 6.5

Data Structures After Building Nets

no provision to suspend blocks in mid-air, we begin construction of the goal state from the ground (bottom of the semantic net) upward as we do with most other physical construction (e.g., cars, houses, etc.).

As such, an updating of the Where-net will be attempted to reflect the achieving of the subgoal (is-on b c) from the Find-net. This update causes an intervention from the critics gallery, namely Critic-1. If the update (is-on b c) is allowed in the Where-net, then the subgoal (is-on a b) will be impeded. This could be reasoned from the following facts (possible implementation scheme for Critic-1):

- to establish (is-on a b), the indegree of node a must be zero (i.e., (is-clear a) is true), and the same must be true for node b.

- the path length from node a to the root was 1 and now it is 2 which impedes a prerequisite of (is-clear a) if (is-on a b) can be achieved.

Let us see how these conditions can be "reasoned". The prerequisites for an "is-on" relation exist if both arguments

of the relation are on level 1 of the "Level-SNet1" and are both contained in an "is-clear" relation. We see in Figure 6.5 that such is the case. So then, the subgoal (is-on b c) can be accomplished.

For the next condition, we find what level node a is on by LEVELNUM(a). Then it is determined if node a is on the path of node c. This is also determined by searching the list "Where-net(c)". From these conditions Critic-1 prevents the action (is-on b c) from being accomplished. With that being the case, line 14 of REASON is invoked and GENERATE puts a prerequisite for a subgoal on the Hold-List. Since (is-on b c) has its prerequisites already established, the prerequisites for (is-on a b) is attempted to be determined.

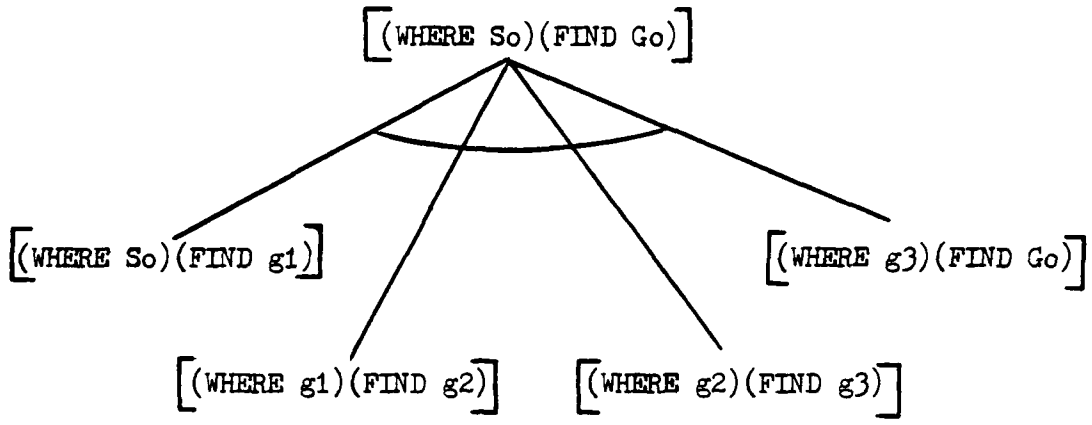
If no prerequisite can be determined within the structural hierarchy of the Critics Gallery, then SEMANTIC-SEC is unable to provide a sequence. But in this case, however, it is easily determined that the prerequisite needed is to clear block a.

Here is a fine point: Since the prerequisite does not exists, it therefore must be accomplished so the reasoning language has to direct the target language to achieve it. Therefore, the prerequisite is generated in terms of target language instructions. Importantly, other conditions that are needed must remain invariant. For example, when the reasoning language determines that the prerequisite to clear block a is required, several things must be accomplished. First, the subgoal to clear block a must be generated. Then the already

established prerequisites for other subgoals, namely (is-clear b) and (is-clear c) must be maintained, must not be violated. Therefore, the subgoal to clear block a is constrained by adding (i.e., logical AND) the established prerequisites of the other subgoals.

The results are that we can see that adequate reasoning can be accomplished by using the semantic net. We see in lines 15 and 16 that the structures are updated with the prerequisite and invariants.

REASON returns control back to SEMANTIC-SEQ where line 6 is executed. There the output function is invoked and the sequence of specifications is output. Figure 6.6 depicts it in the AND/OR graph form.

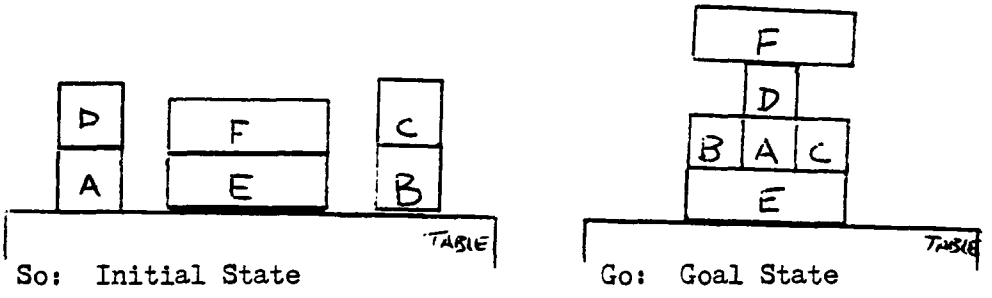


So = ((is-on c a)(is-clear b)(is-clear c))
Go = ((is-on a b)(is-on b c))
g1 = ((clear a)(is-clear c)(is-clear b))
g2 = ((is-on b c))
g3 = ((is-on a b))

Figure 6.6
Sequence of Subspecifications for Sussman's Anomaly

F. SUSSMAN'S SHOWING-OFF

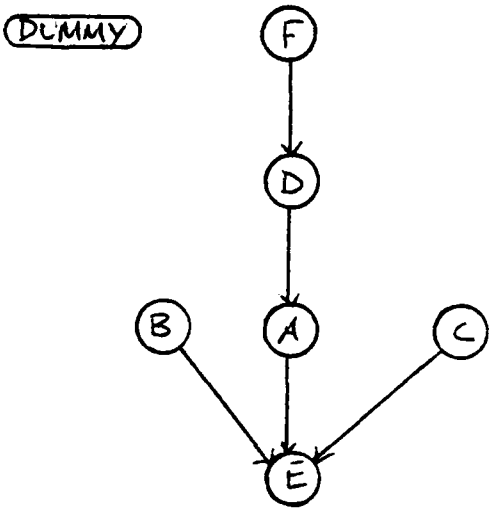
This example is also taken from [13] of Sussman's. We present it because it shows the ease of updating these methods to expand the power of the synthesis system over a problem domain, and it shows how the meaning of the interpretation given to the possible connections of the semantic net can affect a specific problem domain. Figure 6.7 shows the initial state, goal state and the procedural specification for Showing-Off.




```
(WHERE (is-on d a)(is-on f e)(is-on c b))
        (FIND (is-on d a)(is-on b e)
              (is-on a e)(is-on c e)
                    (is-on f d))
```

Figure 6.7
Showing-Off

The point of interest in this example is quickly seen when we observe the Find-net, which is shown in Figure 6.8. We see that the terminal node, block E, has an indegree of 3. What does this mean to the synthesis system as it currently exist (i.e., present set of defined operators, critics, etc.)?



Find-net

Figure 6.8
Find-net for Showing-Off

Recall that the rules of the Blocks World provides the restriction that no two objects can occupy the same space during the same time. How is the enforcement of this rule ascertained? It now becomes obvious to us that the current synthesis system for the Blocks World can not accommodate problems where any indegree is greater than 1. Likewise, the interpretation given to an indegree greater than 1 is important. If we interpret such a condition to mean that blocks are occupying the same space, then a simple critic can be added to the critics gallery that would simply disallow such a problem. That is, the synthesis system would output that no solution is possible. Otherwise, there is the implication that blocks are of different sizes; this interpretation allows a variety of possibilities within the problem domain.

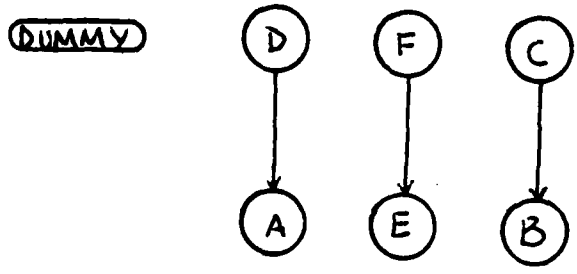
For different size blocks, the synthesis system requires more knowledge in its reasoning language (e.g., know size of a block, determine available space, etc.). This additional knowledge will result in adding other relational operators to reason this information. Similar operators are added to the target language to rearrange (e.g., shift-over, make-room, etc.) blocks on top of some block, or for a block on top of many blocks.

Let us assume then that we have the additional operators and critics. Particularly, we have a critic, critic-4, that concerns itself with the multiple indegree of a node. Also, we

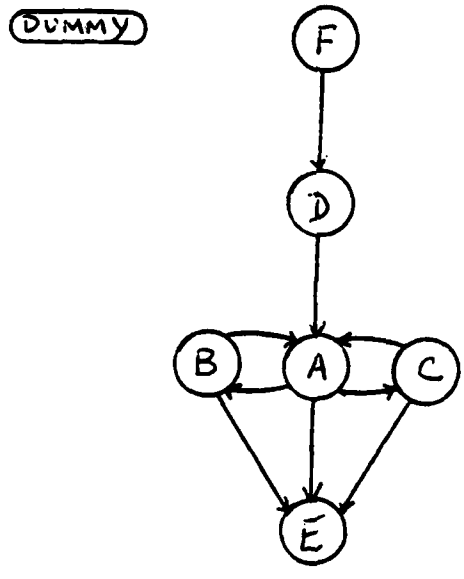
have the relational "is-room" that determines if space is available for block a on some block b. That is, (is-room a b) is TRUE if there exists on block b room to place block a. Otherwise, (is-room a b) is FALSE. Because we are concerned with the "exact" positioning of blocks on a single block, we provide the relational "is-next2" that means block a is adjacent to and along the side of block b (i.e., (is-next2 a b)). Furthermore, we note that is-next2 is a reflexive relational property. That is, (is-next2 a b) if and only if (is-next2 b a). Consequently, the use of is-next2 causes a loop in the semantic net.

Before invoking SEMANTIC-SEQ with the specification that defines Showing-Off, we augment the specification in the Find constraint with the relationships (is-next2 b a), (is-next2 a b), (is-next2 a c) and (is-next2 c a). Figure 6.9 depicts the Where-net and the Find-net for the modified specification of Showing-Off. Note the loop created on level 2 of the Find-net.

As REASON begins to "reason" the sequence of specifications, let us note the more interesting aspects of the process. When a level 3 task is selected, critic-4 will make known the fact that node e has multiple indegree. This then causes all is-on relations coming into node e to be located (e.g., find all level 3 task with is-on where arg2 of is-on is node e). When the is-on relationships are known the is-room relation is then applied. For example, it might appear as:



Where-net



Find-net

Figure 6.9
Where-net and Modified Find-net Using Is-next2

```
is-room:  If (size(a) + size(b) + . . . + size(n))
                        less than or equal
              size(e)
          then
              is-room  <--- TRUE
          else
              is-room  <--- False;
          return(is-room);
```

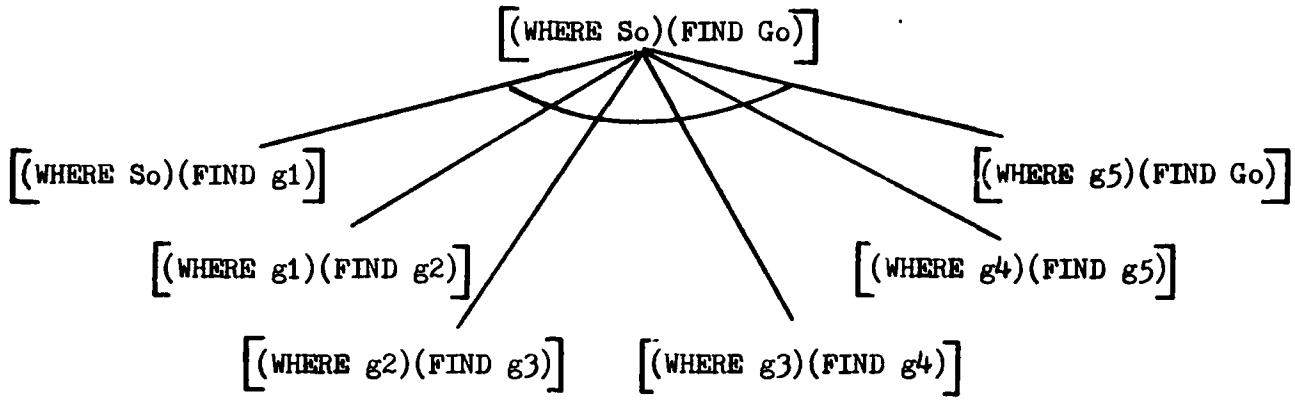
If is-room returns with the value False then REASON knows that it is not possible to place all the blocks on block e, and REASON would invoke an error-handler. If is-room returns TRUE the process continues. We assume is-room is TRUE. Recall, however, the relation is-next2 causes a loop. Therefore, the nodes b, a and c are considered as a single entity. That means that the is-next2 relationships are all considered "en masse".

Let us consider the affect of the "en masse" treatment of nodes a, b and c upon level 3 tasking. Since all the is-on relationships of level 3 are involved in the "en masse" nodes, the control structure does not allow any level 3 task to be accomplished until REASON (or some other mechanism) can achieve the effects caused by the loop. Assume that the ordering of the nodes a, b and c are accomplished. This then supplies the ordering for placing blocks onto block e. REASON is now at the point of trying to establish if the blocks b, a and c, in that order, can be placed on block e.

For each is-on of level 3, the Where-net is considered to determine if the appropriate prerequisites exist. This being done, the level 3 tasking causes the following to be placed on the start of the Find-net:

((clear b) (is-clear c) (clear e) (clear a))

When this has been done, REASON attempts to reason the next level of tasking, namely level 2 tasking. The Where-net is examined for the existence of the prerequisites for the level 2 tasking and would find that some of the tasking is already achieved. Recall, some of level 2 tasking was implicitly achieved because of the "en masse" consideration given to the loop on level 2. This consideration postponed level 3 tasking, but it also caused level 3 tasking to be achieved in an ordered sequence that partially achieved level 2 tasking. That is, blocks a, b and c were placed in their specified order. The remaining level 2 task is (is-on d a). As a result of clearing blocks a, e and b, the block d is not necessarily "cleared". However, from the previous examples we know that SEMANTIC-SEQ can sequence the remaining tasks of Showing-Off. The sequence of specifications for Showing-Off is shown in Figure 6.10.



So = ((is-on d a)(is-on f e)(is-on c b))

Go = ((is-on d a)(is-on b e)(is-on a e)(is-on c e)(is-on f d)(is-next2 b a)(is-next2 a b)
(is-next2 a c)(is-next2 c a))

g1 = ((clear b)(is-clear c)(clear e)(clear a))

g2 = ((is-next2 b a)(is-next2 a b)(is-next2 a c)(is-next2 c a))

g3 = ((is-on b e)(is-on a e)(is-on c e))

g4 = ((is-on d a))

g5 = ((is-on f d))

Figure 6.10

Sequence of Subspecifications for Showing-Off

VII. OTHER PROBLEM DOMAINS

A. GENERAL

It is desirable that the methods used to determine a sequence of specifications be as general as possible. That is, we want SEMANTIC-SEQ to be applicable for as many problem domains as is possible. Currently, we do not know how general SEMANTIC-SEQ is.

However, there is this encouraging aspect about SEMANTIC-SEQ's scope of applicability. We know that n-ary relations can be easily converted to binary relations. Consequently, the arity of a problem domain's relations should not excluded the problem domain from being considered by SEMANTIC-SEQ. That is, since the underlying data structure in SEMANTIC-SEQ is a graph, there are many optimally constructed algorithms for "processing" graph and tree type structures. Also, tree type structrues are commonly used in our everyday problem-solving activities (e.g., decision-making). As a result, we are likely to find a wide variety of domains to which we can apply SEMANTIC-SEQ.

B. NON-BLOCKS WORLD EXAMPLE

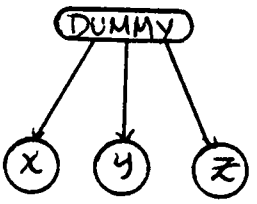
We consider a Non-Blocks World problem to provide insight about the scope of SEMANTIC-SEQ. In so doing, it is hoped that inadequacies and possible extensions can be realized for the

purpose of increasing SEMANTIC-SEQ's power. Therefore, consider the problem of sorting some list of numbers and finding some i-th element of the sort. We define Sort: List ---> list, where list is the set of list of natural numbers. (Sort x y) means that the list x is the input and y is the sorted list after achieving Sort. Further, y2 is the second element of the well-ordering of the sorted list y. A description of Sort is:

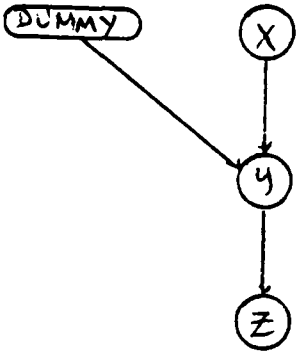
[(WHERE(x and y are list of natural numbers))
(FIND(is-perm x y)(is-ordered y))]

The term (is-perm L1 L2) determines if the list L1 and the list L2 represent a permutation of each other. Is-perm returns the value TRUE if L1 is a permutation of L2. Otherwise, is-perm returns the value FALSE. The is-ordered relation determines the well-ordering property of some list of elements.

Therefore, given a specification to the problem of sorting and finding the i-th value, we would expect SEMANTIC-SEQ to build the following structures shown in Figure 7.1.



Where-net



Find-net

Figure 7.1
Sort and Find Example

We assume that the structures shown in Figure 7.1 are built. In which direction does REASON begin to act upon the Find-net? What are the prerequisites for the relations? We begin to see that the user must provide SEMANTIC-SEQ with the information to perform reasoning.

However, let us continue with the problem. Our intuition tells us that we should process the Find-net from top to bottom. That is, we achieve is-ordered and then achieve is-equal. Intuitively, we feel that this may be a feasible approach to problems in this domain (i.e., start from the top of the net). But is this approach valid for all problems in the domain?

Before addressing the issues that have been raised, let us consider a slightly more complicated example. We would like to sort a list of numbers and then find the median element of the sorted list. What makes this example interesting is the fact that the computation of the median depends on the input list having an odd or even number of elements. For instance, if the length of the list is odd then the median is FLOOR [(length / 2) + 1)] element of the sorted list. Otherwise, it is the average of (length / 2) element + (length /2 + 1) element. Therefore, suppose the specification of such a problem was:

```
[(WHERE( x and y are list of natural numbers)
        ( z is real number))
              (FIND(is-perm x y)(is-ordered y)
                                (is-median y z))]
```

We can easily see that structures similar to Figure 7.1 would be built from such a specification. Furthermore, it becomes obvious to us that SEMANTIC-SEC must have the capability to know which type median, odd or even, the target language must compute. In short, the task-specific knowledge for such a domain must provide the necessary structure and relational test. Specifically, CONFLICTS and GENERATE must provide this information. For example, when the prerequisites for the operator is-median is being determine, the condition that its oddness be known is required before it can be

included in the Where-net. For instance, a relation is-odd(y) could be made where y is the list from which the median element is to be determined. Based on that test, the appropriate formula is then selected and added to the prereq-list in GENERATE. Again, when we consider the relations involved in the sorting, likewise prerequisite test can be made and structured to produce the appropriate commands to the target language.

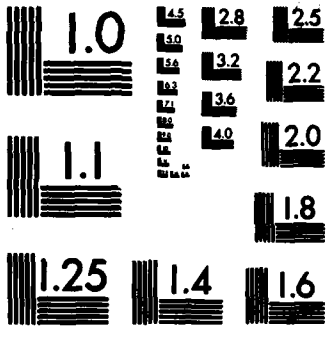
This brings us to the matter of the primitive operators that are available to the reasoning language. Recall Smith in [6] defines a top-down program synthesis system. If we can accept Smith's structure and if the problem domain can be defined in such an environment, then much of the information that SEMANTIC-SEQ requires would already exist in the top-down system.

C. INADEQUACIES AND EXTENSIONS

SEMANTIC-SEQ has the following inadequacies. First, the user is burden with supplying very detailed information about his problem domain. For example, he must know the special critics for each relation. Secondly, the choice of prerequisites is dependent upon the primitives in the reasoning language. Further, SEMANTIC-SEQ provides only one level of detail. That is, the system NOAH in [11] is able to provide varying levels of detail about any subgoal until primitive level events for that subgoal are reached.

AD-A127 417 A HEURISTIC FOR DECOMPOSING A PROBLEM INTO A SEQUENCE OF SUBPROBLEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D V EVANS DEC 82 2/2

UNCLASSIFIED F/G 9/2 NL



SEMANTIC-SEQ, on the other hand, can only provide the same level of detail about a subgoal as was provided it on input of the specification. SEMANTIC-SEQ does not provide any information when the entire Find constraint of the specification results in a loop.

Additionally, the characteristics surrounding the ordering or formatting of arguments within a relation is not fully understood. For instance, which argument should be listed as the first argument of the relation? Let us again consider the example problem of sorting and finding the median element. In particular, consider the relation (is-median y z). If the arguments had been reversed (i.e., (is-median z y)), then all subgoals of the example problem would have resided on the same level of the Find-net. This also would have prevented SEMANTIC-SEQ from providing a sequence of subspecifications. Consequently, SEMANTIC-SEQ is currently dependent upon the arguments of relations being ordered in a manner meaningful to the problem domain. This ordering of arguments is the concept of "flow" or "from input argument to output argument" that was developed in Chapter V. The requirement of such a dependency is also unknown at this time.

The most obvious extension for SEMANTIC-SEQ is to provide the capability of converting a n-ary relation to a binary relation. Such an extension would assist in determining the scope of SEMANTIC-SEQ in other problem domains. Also, it would assist in the determining what affect, if any, relational

properties like antireflexive, symmetry, etc. have on the semantic net regarding a sequence of specifications. In short, any extension should be aimed at helping to determine the scope of SEMANTIC-SEQ.

VIII. SUMMARY

This research has shown that it is possible to extract information from the problem specification and determine a sequence of specifications using semantic networks that provides a solution to the problem. To that extent, we consider this endeavor a success. The method we have presented is considered novel because it considers the problem specification "in toto" in its attempt to determine a sequence of specifications.

The motivation for such an approach is the belief that the idea of how to solve a problem is somehow contained in a problem's specification. As we become adept at discovering the artificial intelligence centered about "ideas", mechanisms more general than SEMANTIC-SEQ can be realized.

The encouraging facts about SEMANTIC-SEQ in view of its inadequacies as stated in the previous chapter are as follows. First, the prerequisite information that SEMANTIC-SEQ needs will largely be the contained in the information needed to define a Top-Down Program Synthesis System. Smith in [6] presents a structure for top-down program synthesis. Therefore, if the problem domain can be defined in terms of such a structure, then much of the information required by SEMANTIC-SEQ already will have been defined.

Secondly, SEMANTIC-SEQ is considered a cheap approach to determining sequence because it uses much of the same information used to define the problem domain. Also, the structure of SEMANTIC-SEQ is simple. The criticism approach provides the advantages of modularity, isolates the concern of operator action, and allows for easy expansion of capabilities to the problem domain because critics can be structured hierarchically.

LIST OF REFERENCES

1. Simon, H. A., "Artificial Intelligence Research Strategies in the Light of AI Models of Scientific Discovery," Naval Research Reviews 34 No. 2 (1982): pp. 2 - 15.

2. Nilsson, N. J., Problem-Solving Methods in Artificial Intelligence, p. vii and 7, McGraw-Hill, Inc., 1971.

3. Sprague, R. H. and Carlson, E. D., Building Effective Decision Support Systems, 26 - 27, Prentice-Hall, Inc., 1982.

4. Yasaki, E. K., "Tokyo Looks to the '90s," Datamation, January 1982, pp. 110 - 115.

5. Manna, Z. and Waldinger, R., "Synthesis: Dreams ---> Programs," IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, pp. 294 - 328, July 1979.

6. Naval Postgraduate School Report NPS52-82-011, Top-Down Synthesis of Simple Divide and Conquer Algorithms, by D. R. Smith, pp. 30 - 33 and 3, November 1982.

7. Liskov, B. H. and Berzins, V., "An Appraisal of Program Specifications," in Research Directions in Software Technology, pp. 276 - 301. Edited Peter Wegner, Associate Editors: Jack Dennis, Michael Hamrer and D. Teichroew. Cambridge, Massachusetts: The MIT Press, 1979.

8. Waldinger, R., "Achieving Several Goals Simultaneously," in Readings in Artificial Intelligence, pp. 250 - 271. Edited Bonnie Lynn Webber and Nils J. Nilsson. Palo Alto, Califorina: Tioga Publishing Co., 1981.

9. Barstow, D. R., "An Experiment in Knowledge-based Automatic Programming," in Readings in Artificial Intelligence, pp. 289 - 312.

10. Manna, Z. and Waldinger, R., "A Deductive Approach to Program Synthesis," in Readings in Artificial Intelligence, pp. 141 - 172.

11. Sacerdoti, E. D., A Structure for Plans and Behavior, New York: American Elsevier Publishing Co., 1977.

12. Idem., "The Nonlinear Nature of Plans," in Proceeding of 4th International Joint Conference Artificial Intelligence, Tbilisi, USSR, September 1975, pp. 206 - 214.

13. Sussman, G. J., A Computer Model of Skill Acquisition, New York: American Elsevier Publishing Co., 1975.

14. Nilsson, N. J., Principles of Artificial Intelligence, pp. 361 - 411, Palo Alto, California: Tioga Publishing Co., 1980.

15 Fikes, R. E., Hart, P. E., and Nilsson, N. J., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol 3, pp. 251 - 288, 1972.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Assistant Professor Douglas R. Smith, Code 52Sc Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	Assistant Professor Bruce J. MacLennan, Code 52M1 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Associate Professor Harold Fredricksen, Code 53Fs Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
7.	MCS Development Office (ATTN: Mr. William Snook) Marine Corps Logistics Base Albany, Georgia 31704	1
8.	Captain Donald V. Evans, USMC 391-C Ricketts Road Monterey, California 93940	6

9. Marine Corps Representative, He-E329 1
Naval Postgraduate School
Monterey, Califcrnia 93940

END

FILMED

5-83

DTIC